**Отчет по лабораторной работе № 24** по курсу “ Практикум на ЭВМ ”

Студент группы М8О-102Б-21, Яценко Александр Владимирович, № по списку 20

Контакты: e-mail - alexander.iatsenko@gmail.com

Работа выполнена: « » \_\_\_\_\_\_\_\_\_\_\_\_201\_\_\_г.

Преподаватель: Никулин Сергей Петрович Каф.806\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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

Отчет сдан « » \_\_\_\_\_\_\_\_\_201 \_\_ г., итоговая оценка \_\_\_\_\_

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

**1. Тема:** Алгоритмы и структуры данных.

**2. Цель работы:** составить программу на языке Си, производящую преобразования арифметических выражений с применением деревьев.

**3. Задание (**задача 25**):** Разложить выражение a^(b+c) в произведение a^b\*a^c.

**4. Оборудование:**

*Оборудование ПЭВМ студента:*

Процессор: AMD Ryzen 5 4600H, с ОП 6 Мб (виртуальная машина), НМД 25600 Мб. Монитор: Huawei 16,1 IPS 1920×1080, 137 PP.

**5. Программное обеспечение ЭВМ студента, если использовалось:**

Операционная система семейства Linux, наименование Fedora, версия 5.13.12-200. Fc24.x86\_64 GNU/Linux

Прикладные системы и программы: CLion, emacs

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

**Тесты программы:**

**Ввод:**

((а+9)

**Результат:**

Некорректное выражение

**Ввод:**

a\*(5-f)^2

**Результат:**

a\*(5-f)^2

**Ввод:**

a^(b+c)

**Результат:**

a^b\*a^c

**Ввод:**

a^((b+c)+d)

**Результат:**

a^b\*a^c\*a^d

**Ввод:**

a-7\*f\*t+(b-g)^(2+2)\*c

**Результат:**

a-7\*f\*t+(b-g)^ 2\*(b-g)^ 2\*c

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

**Код программы:**

#include <stdio.h>  
#include <stdlib.h>  
#include <stdbool.h>  
  
struct stack rev;  
struct stack \*reverse = &rev;  
  
int depth = 0;  
  
struct element  
{  
 char data;  
 struct element \*prev;  
};  
  
typedef struct element \*p\_element;  
  
struct stack  
{  
 int size;  
 struct element \*top;  
};  
  
struct tnode  
{  
 char d;  
 struct tnode \*ld;  
 struct tnode \*rd;  
};  
  
typedef struct tnode \*stnode;  
  
void stack\_create (struct stack \*t)  
{  
 t->top = 0;  
 t->size = 0;  
}  
  
void stack\_destroy (struct stack \*t)  
{  
 struct element \*prev\_save;  
 while(t->size > 0)  
 {  
 prev\_save = t->top->prev;  
 free(t->top);  
 t->top = prev\_save;  
 t->size--;  
 }  
 t->top = NULL;  
}  
  
void stack\_push (struct stack \*t, char c)  
{  
 p\_element new = (struct element\*) malloc(sizeof(struct element));  
 new->data = c;  
 new->prev = t->top;  
 t->top = new;  
 t->size++;  
}  
  
bool stack\_pop (struct stack \*t)  
{  
 struct element \*prev\_save;  
 if (t->size == 0)  
 return(false);  
 else  
 {  
 prev\_save = t->top->prev;  
 free(t->top);  
 t->top = prev\_save;  
 t->size--;  
 return(true);  
 }  
}  
  
char stack\_top\_element (struct stack \*t)  
{  
 if(t->top)  
 return(t->top->data);  
}  
  
bool is\_number (char c)  
{  
 if ((c >= 'a' & c <= 'z') || ('0' <= c & c <= '9') || ('A' <= c & c <= 'Z'))  
 return(true);  
 else  
 return(false);  
}  
  
bool is\_opeator (char c)  
{  
 if (c == '+' || c == '-' || c == '\*' || c == '/' || c == '^')  
 return(true);  
 else  
 return(false);  
}  
  
int check\_prioity (char c)  
{  
 switch(c)  
 {  
 case('+'):  
 return(1);  
  
 case('-'):  
 return(1);  
  
 case('\*'):  
 return(2);  
  
 case('/'):  
 return(2);  
  
 case('^'):  
 return(3);  
 }  
}  
  
int convert(char gg[])  
{  
 char c;  
 struct stack st;  
 struct stack \*t = &st;  
 stack\_create(t);  
 stack\_create(reverse);  
 for(int i = 0; (c = gg[i]) != '\0'; i++)  
 {  
 if(is\_number(c))  
 stack\_push(reverse, c);  
  
 if(is\_opeator(c))  
 {  
 while(is\_opeator(stack\_top\_element(t)))  
 {  
 if(check\_prioity(stack\_top\_element(t)) >= check\_prioity(c))  
 {  
 stack\_push(reverse, stack\_top\_element(t));  
 stack\_pop(t);  
 }  
 else  
 break;  
 }  
 stack\_push(t, c);  
 }  
 if(c == '(')  
 stack\_push(t, c);  
 if(c == ')')  
 {  
 while(stack\_top\_element(t) != '(')  
 {  
 if(t->size == 0)  
 {  
 printf("Некорректное выражение\n");  
 exit(EXIT\_SUCCESS);  
 }  
 stack\_push(reverse, stack\_top\_element(t));  
 stack\_pop(t);  
 }  
 stack\_pop(t);  
 }  
 }  
 while(t->size > 0)  
 {  
 if (t->top->data == '(')  
 {  
 printf("Некорректное выражение\n");  
 exit(EXIT\_SUCCESS);  
 }  
 stack\_push(reverse, stack\_top\_element(t));  
 stack\_pop(t);  
 }  
 stack\_destroy(t);  
}  
  
void treeprint(stnode tree, int guidelines)  
{  
 if (tree != NULL)  
 {  
 depth += 4;  
 treeprint(tree->rd, 1);  
 for (int i = 0; i < depth; i++)  
 printf(" ");  
 switch(guidelines)  
 {  
 case 1:  
 {  
 printf("/%c\n", tree->d);  
 break;  
 }  
 case 2:  
 {  
 printf("\\%c\n", tree->d);  
 break;  
 }  
 case 3:  
 {  
 printf("%c\n", tree->d);  
 break;  
 }  
 }  
 treeprint(tree->ld, 2);  
 depth -= 4;  
 }  
}  
  
stnode to\_tree(char c)  
{  
 char c\_new;  
 stnode node;  
 node = (stnode) malloc(sizeof(stnode));  
  
 node->d = c;  
 if(is\_number(c))  
 {  
 node->rd = NULL;  
 node->ld = NULL;  
 return(node);  
 }  
 c\_new = reverse->top->data;  
 stack\_pop(reverse);  
 node->rd = to\_tree(c\_new);  
 c\_new = reverse->top->data;  
 stack\_pop(reverse);  
 node->ld = to\_tree(c\_new);  
 return(node);  
}  
  
stnode tree\_copy (stnode tree)  
{  
 if(tree == NULL)  
 return(NULL);  
 stnode node;  
 node = (stnode) malloc(sizeof(stnode));  
  
 node->d = tree->d;  
 node->rd = tree\_copy(tree->rd);  
 node->ld = tree\_copy(tree->ld);  
 return(node);  
}  
  
stnode tree\_replace (stnode tree)  
{  
 stnode a\_save, a\_copy, b\_save, c\_save, node\_mult, node\_power\_l, node\_power\_r;  
  
 node\_mult = (stnode) malloc(sizeof(stnode));  
 node\_power\_l = (stnode) malloc(sizeof(stnode));  
 node\_power\_r = (stnode) malloc(sizeof(stnode));  
  
 a\_save = tree->ld;  
 b\_save = tree->rd->ld;  
 c\_save = tree->rd->rd;  
 a\_copy = tree\_copy(a\_save);  
  
 node\_mult->d = '\*';  
 node\_mult->ld = node\_power\_l;  
 node\_mult->rd = node\_power\_r;  
  
 node\_power\_l->d = '^';  
 node\_power\_l->ld = a\_save;  
 node\_power\_l->rd = b\_save;  
  
 node\_power\_r->d = '^';  
 node\_power\_r->ld = a\_copy;  
 node\_power\_r->rd = c\_save;  
  
 free(tree->rd);  
 free(tree);  
  
 return(node\_mult);  
}  
  
stnode tree\_search (stnode tree)  
{  
 if (tree != NULL)  
 {  
 if (tree->d == '^')  
 if (tree->rd->d == '+')  
 {  
 tree = tree\_replace(tree);  
 tree = tree\_search(tree);  
 return(tree);  
 }  
 tree->ld = tree\_search(tree->ld);  
 tree->rd = tree\_search(tree->rd);  
 return(tree);  
 }  
 else  
 {  
 return(NULL);  
 }  
}  
  
void print\_expression(stnode tree)  
{  
 if (tree == NULL)  
 {  
 return;  
 }  
 if (is\_opeator(tree->d) && is\_opeator(tree->ld->d) && (check\_prioity(tree->ld->d) < check\_prioity(tree->d) || tree->ld->d == '^' && tree->d == '^'))  
 {  
 printf("(");  
 print\_expression(tree->ld);  
 printf(")");  
 }  
 else  
 {  
 print\_expression(tree->ld);  
 }  
  
 printf("%c", tree->d);  
  
 if (is\_opeator(tree->d) && is\_opeator(tree->rd->d) && (check\_prioity(tree->rd->d) < check\_prioity(tree->d) || tree->rd->d == '^' && tree->d == '^'))  
 {  
 printf("(");  
 print\_expression(tree->rd);  
 printf(")");  
 }  
 else  
 {  
 print\_expression(tree->rd);  
 }  
}  
  
int main()  
{  
 int option, control = 1;  
 char c\_first, input\_string[1000];  
 stnode root;  
  
 while (control!= 0)  
 {  
 printf("1. Создать дерево\t 2. Выполнить преобразования\t 3. Распечатать дерево\t 4. Распечатать выражение\t 5. Выход\n");  
 scanf("%d", &option);  
 switch (option)  
 {  
 case 1:  
 {  
 printf("Введите выражение\n");  
 scanf("%s", input\_string);  
 convert(input\_string);  
 c\_first = reverse->top->data;  
 stack\_pop(reverse);  
 root = to\_tree(c\_first);  
 break;  
 }  
  
 case 2:  
 {  
 root = tree\_search(root);  
 break;  
 }  
  
 case 3:  
 {  
 treeprint(root, 3);  
 break;  
 }  
  
 case 4:  
 {  
 print\_expression(root);  
 printf("\n");  
 break;  
 }  
  
 case 5:  
 {  
 control = 0;  
 break;  
 }  
  
 default:  
 {  
 printf("Неверный выбор пункта меню\n");  
 break;  
 }  
 }  
 }  
  
 stack\_destroy(reverse);  
}

*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,подписанныйпреподавателем)

1. Создать дерево 2. Выполнить преобразования 3. Распечатать дерево 4. Распечатать выражение 5.Выход

1

Введите выражение

((а+9)

Некорректное выражение

1. Создать дерево 2. Выполнить преобразования 3. Распечатать дерево 4. Распечатать выражение 5.Выход

1

Введите выражение

a\*(5-f)^2

1. Создать дерево 2. Выполнить преобразования 3. Распечатать дерево 4. Распечатать выражениe 5.Выход

3

/2

/^

/f

\-

\5

\*

\a

1. Создать дерево 2. Выполнить преобразования 3. Распечатать дерево 4. Распечатать выражение

5.Выход

2

1. Создать дерево 2. Выполнить преобразования 3. Распечатать дерево 4. Распечатать выражение

5.Выход

4

a\*(5-f)^2

1. Создать дерево 2. Выполнить преобразования 3. Распечатать дерево 4. Распечатать выражение

5.Выход

5

1. Создать дерево 2. Выполнить преобразования 3. Распечатать дерево 4. Распечатать выражение

5.Выход

1

Введите выражение

a^(b+c)

1. Создать дерево 2. Выполнить преобразования 3. Распечатать дерево 4. Распечатать выражение 5.Выход

3

/c

/+

\b

^

\a

1. Создать дерево 2. Выполнить преобразования 3. Распечатать дерево 4. Распечатать выражение 5.Выход

2

1. Создать дерево 2. Выполнить преобразования 3. Распечатать дерево 4. Распечатать выражение 5.Выход

3

/c

/^

\a

\*

/b

\^

\a

1. Создать дерево 2. Выполнить преобразования 3. Распечатать дерево 4. Распечатать выражение 5.Выход

4

a^b\*a^c

1. Создать дерево 2. Выполнить преобразования 3. Распечатать дерево 4. Распечатать выражение 5.Выход

5

1. Создать дерево 2. Выполнить преобразования 3. Распечатать дерево 4. Распечатать выражение 5.Выход

1

Введите выражение

a^((b+c)+d)

1. Создать дерево 2. Выполнить преобразования 3. Распечатать дерево 4. Распечатать выражение 5.Выход

3

/d

/+

/c

\+

\b

^

\a

1. Создать дерево 2. Выполнить преобразования 3. Распечатать дерево 4. Распечатать выражение 5.Выход

2

1. Создать дерево 2. Выполнить преобразования 3. Распечатать дерево 4. Распечатать выражение 5.Выход

3

/d

/^

\a

\*

/c

/^

\a

\\*

/b

\^

\a

1. Создать дерево 2. Выполнить преобразования 3. Распечатать дерево 4. Распечатать выражение 5.Выход

4

a^b\*a^c\*a^d

1. Создать дерево 2. Выполнить преобразования 3. Распечатать дерево 4. Распечатать выражение 5.Выход

5

1. Создать дерево 2. Выполнить преобразования 3. Распечатать дерево 4. Распечатать выражение 5.Выход

1

Введите выражение

a-7\*f\*t+(b-g)^(2+2)\*c

1. Создать дерево 2. Выполнить преобразования 3. Распечатать дерево 4. Распечатать выражение 5.Выход

3

/c

/\*

/2

/+

\2

\^

/g

\-

\b

+

/t

/\*

/f

\\*

\7

\-

\a

1. Создать дерево 2. Выполнить преобразования 3. Распечатать дерево 4. Распечатать выражение 5.Выход

2

1. Создать дерево 2. Выполнить преобразования 3. Распечатать дерево 4. Распечатать выражение 5.Выход

3

/c

/\*

/2

/^

/g

\-

\b

\\*

/2

\^

/g

\-

\b

+

/t

/\*

/f

\\*

\7

\-

\a

1. Создать дерево 2. Выполнить преобразования 3. Распечатать дерево 4. Распечатать выражение 5.Выход

4

a-7\*f\*t+(b-g)^2\*(b-g)^2\*c

1. Создать дерево 2. Выполнить преобразования 3. Распечатать дерево 4. Распечатать выражение 5.Выход

5

1. **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. | Дата | Время | Событие | Действие по исправлению | Примечание |
|  | или |  |  |  |  |  |
|  | дом. |  |  |  |  |  |
|  |  |  |  |  |  |  |

1. **Замечания автора** по существу работы: -
2. **Выводы:** работа выполнена

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