|  |  |
| --- | --- |
| + | **Отчёт по лабораторной работе** №24  по курсу 1 фундаментальная информатика  студента группы М8О-105Б-21 Бондаревой Елены Евгеньвны, № по списку 1  Контакты www, e-mail, icq, skype : lena\_bondareva\_03@mail.ru  Работа выполнена: «10» мая 2022г.  Преподаватель: каф.806 В.К.Титов  Входной контроль знаний с оценкой  Отчёт сдан, итоговая оценка  Подпись преподавателя |

1. **Тема**: «Дерево выражений».
2. **Цель работы**: Составить программу выполнения заданных преобразований арифметических выражений с применением деревьев. Преобразование выражения в дерево рекомендуется осуществлять одним из известных методов. Преобразование выражения реализовать в виде набора подпрограмм. Программа должна вводить и печатать выражения в исходном виде, преобразовывать их в деревья, выполнять заданное действие и печатать результат в виде дерева и в текстовом представлении.
3. **Задание:**

Убрать из выражений все сомножители, равные единицы.

1. **Оборудование** (*лабораторное*):

ЭВМ -, процессор -, имя узла сети-с ОП -МБ

НМД -ГБ. Терминал - адрес -. Принтер -.

Другие устройства -.

*Оборудование ПЭВМ студента, если использовалось:*

Процессор Intel(R) Core(TM) i3-7020U CPU @ 2.30GHz , ОП 6 ГБ, НМД 240 ГБ. Монитор IPS 1920x1080

Другие устройства -.

1. **Программное обеспечение** (*лабораторное*):

Операционная система семейства -, наименование - версия -

Интерпретатор команд - версия -

Система программирования -версия -

Редактор текстов - версия -

Утилиты операционной системы -

Прикладные системы и программы -

Местонахождения и имена файлов программ и данных-

*Программное обеспечение ЭВМ студента, если использовалось:*

Операционная система семейства UNIX/GNU , наименование Ubuntu версия x86\_64

Интерпретатор команд bash

Редактор текстов emax

Утилиты операционной системы: head, du, grep, sum, tee, file, find, diff, tail, od, wc, cut, tar, touch, paste, uniq, gzip, sort, cmp, bzip2.

Прикладные системы и программы VTM-diagram

Местонахождения и имена файлов программ и данных -

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

Необходимо убрать из выражений все сомножители, равные единицы. Вначале напишем функции, каждая из которых будет выполнять строго определенное действие. Например, вывод и печать выражения в исходном виде; преобразование их в деревья; исключить из выражения сомножители, равные 1; печать результата в текстовом представлении и в виде дерева. Основная функция, отвечающая за выполнение заданного действия, *transtree:* как только встречается «\*» происходит проверка множителей, к которому относится данный символ (есть ли среди них единица или нет), то есть осуществляется нахождение единицы. Как только она встретилась, удаляем ее вместе с «\*».

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

#define \_CRT\_SECURE\_NO\_WARNINGS

#include<stdio.h>

#include<stdlib.h>

//#include<conio.h>

typedef char tdata;

int i; char ch;

struct node;

typedef node\* link;

struct node

{

tdata data;

link left, right;

} \*tree;

void printtree(link t)

{

static int l = 0;

l++;

if (t)

{

printtree(t->right);

for (i = 0; i < l; i++)printf(" ");

printf("\\\_\_%c\n", t->data);

printtree(t->left);

}

l--;

} // printtree----------------------------------

int isAN()

{

return (ch >= 'a') && (ch <= 'z') || (ch >= '0') && (ch <= '9');

}

int isN(char c)

{

return (c >= '0') && (c <= '9');

}

link mknode(char c, link l, link r)

{

link t = new node;

t->data = c; t->left = l; t->right = r;

return t;

}

link expr();

link fact()

{

link t = NULL;

scanf("%c", &ch);

if (ch == '(') {

t = expr();

if (ch != ')') printf("ERROR: not )\n");

}

else if (isAN()) t = mknode(ch, 0, 0);

else printf("ERROR: not AN\n");

return t;

}

link term()

{

link tm; int done; char ch1;

tm = fact(); done = 0;

while ((ch != '\n') && (!done))

{

scanf("%c", &ch);

if ((ch == '\*') || (ch == '/')) { ch1 = ch; tm = mknode(ch1, tm, fact()); }

else done = 1;

}

return tm;

}

link expr()

{

link ex; int done; char ch1;

ex = term(); done = 0;

while ((ch != '\n') && (!done))

{

if ((ch == '+') || (ch == '-')) { ch1 = ch; ex = mknode(ch1, ex, term()); }

else done = 1;

}

return ex;

}

void tree2expr(link tree)

{

if (tree)

{

if ((tree->data == '+') || (tree->data == '-')) printf("(");

tree2expr(tree->left);

printf("%c", tree->data);

tree2expr(tree->right);

if ((tree->data == '+') || (tree->data == '-')) printf(")");

}

}

link transtree(link tree)

{

if (tree)

{

if (tree->data == '\*')

{

if (tree->left != 0 && tree->left->data == '1') {

link r = tree->right;

free(tree->left);

free(tree);

return r;

}

if (tree->right != 0 && tree->right->data == '1') {

link l = tree->left;

free(tree->right);

free(tree);

return l;

}

}

tree->left = transtree(tree->left);

tree->right = transtree(tree->right);

}

return tree;

}

int main() {

// clrscr();

printf("Input expression:\n");

tree = expr();

printtree(tree);

printf("\n\n-----------------------\n\n");

tree2expr(tree);

i = 1; while (i) { i = 0; tree = transtree(tree); }

printf("\n\n-----------------------\n\n");

printtree(tree);

printf("\n\n-----------------------\n\n");

tree2expr(tree);

printf("\n\n-----------------------\n\n");

return 0;

// getch();

}//main

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

Допущен к выполнению работы. Подпись преподавателя

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

elena@elena-Aspire-A315-53G:~$ cat tit.txt

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

~ ~

~ Лабораторная работа № 24 ~

~ Дерево выражений ~

~ ~

~ Бондарева Елена ~

~ М8О-105Б-21 ~

~ ~

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

elena@elena-Aspire-A315-53G:~$ cat 24.cpp

#define \_CRT\_SECURE\_NO\_WARNINGS

#include<stdio.h>

#include<stdlib.h>

//#include<conio.h>

typedef char tdata;

int i; char ch;

struct node;

typedef node\* link;

struct node

{

tdata data;

link left, right;

} \*tree;

void printtree(link t)

{

static int l = 0;

l++;

if (t)

{

printtree(t->right);

for (i = 0; i < l; i++)printf(" ");

printf("\\\_\_%c\n", t->data);

printtree(t->left);

}

l--;

} // printtree----------------------------------

int isAN()

{

return (ch >= 'a') && (ch <= 'z') || (ch >= '0') && (ch <= '9');

}

int isN(char c)

{

return (c >= '0') && (c <= '9');

}

link mknode(char c, link l, link r)

{

link t = new node;

t->data = c; t->left = l; t->right = r;

return t;

}

link expr();

link fact()

{

link t = NULL;

scanf("%c", &ch);

if (ch == '(') {

t = expr();

if (ch != ')') printf("ERROR: not )\n");

}

else if (isAN()) t = mknode(ch, 0, 0);

else printf("ERROR: not AN\n");

return t;

}

link term()

{

link tm; int done; char ch1;

tm = fact(); done = 0;

while ((ch != '\n') && (!done))

{

scanf("%c", &ch);

if ((ch == '\*') || (ch == '/')) { ch1 = ch; tm = mknode(ch1, tm, fact()); }

else done = 1;

}

return tm;

}

link expr()

{

link ex; int done; char ch1;

ex = term(); done = 0;

while ((ch != '\n') && (!done))

{

if ((ch == '+') || (ch == '-')) { ch1 = ch; ex = mknode(ch1, ex, term()); }

else done = 1;

}

return ex;

}

void tree2expr(link tree)

{

if (tree)

{

if ((tree->data == '+') || (tree->data == '-')) printf("(");

tree2expr(tree->left);

printf("%c", tree->data);

tree2expr(tree->right);

if ((tree->data == '+') || (tree->data == '-')) printf(")");

}

}

link transtree(link tree)

{

if (tree)

{

if (tree->data == '\*')

{

if (tree->left != 0 && tree->left->data == '1') {

link r = tree->right;

free(tree->left);

free(tree);

return r;

}

if (tree->right != 0 && tree->right->data == '1') {

link l = tree->left;

free(tree->right);

free(tree);

return l;

}

}

tree->left = transtree(tree->left);

tree->right = transtree(tree->right);

}

return tree;

}

int main() {

// clrscr();

printf("Input expression:\n");

tree = expr();

printtree(tree);

printf("\n\n-----------------------\n\n");

tree2expr(tree);

i = 1; while (i) { i = 0; tree = transtree(tree); }

printf("\n\n-----------------------\n\n");

printtree(tree);

printf("\n\n-----------------------\n\n");

tree2expr(tree);

printf("\n\n-----------------------\n\n");

return 0;

// getch();

}//main

elena@elena-Aspire-A315-53G:~$ g++ 24.cpp

elena@elena-Aspire-A315-53G:~$ ./a.out

Input expression:

5\*(a+1)+3+a\*1\*2+a\*1

\\_\_1

\\_\_\*

\\_\_a

\\_\_+

\\_\_2

\\_\_\*

\\_\_1

\\_\_\*

\\_\_a

\\_\_+

\\_\_3

\\_\_+

\\_\_1

\\_\_+

\\_\_a

\\_\_\*

\\_\_5

-----------------------

(((5\*(a+1)+3)+a\*1\*2)+a\*1)

-----------------------

\\_\_a

\\_\_+

\\_\_2

\\_\_\*

\\_\_a

\\_\_+

\\_\_3

\\_\_+

\\_\_1

\\_\_+

\\_\_a

\\_\_\*

\\_\_5

-----------------------

(((5\*(a+1)+3)+a\*2)+a)

-----------------------

elena@elena-Aspire-A315-53G:~$ ./a.out

Input expression:

5\*1+4\*3+1\*(3+3)

\\_\_3

\\_\_+

\\_\_3

\\_\_\*

\\_\_1

\\_\_+

\\_\_3

\\_\_\*

\\_\_4

\\_\_+

\\_\_1

\\_\_\*

\\_\_5

-----------------------

((5\*1+4\*3)+1\*(3+3))

-----------------------

\\_\_3

\\_\_+

\\_\_3

\\_\_+

\\_\_3

\\_\_\*

\\_\_4

\\_\_+

\\_\_5

-----------------------

((5+4\*3)+(3+3))

-----------------------

elena@elena-Aspire-A315-53G:~$ ./a.out

Input expression:

5\*1+3+7+7\*2+2\*1\*3+4\*2

\\_\_2

\\_\_\*

\\_\_4

\\_\_+

\\_\_3

\\_\_\*

\\_\_1

\\_\_\*

\\_\_2

\\_\_+

\\_\_2

\\_\_\*

\\_\_7

\\_\_+

\\_\_7

\\_\_+

\\_\_3

\\_\_+

\\_\_1

\\_\_\*

\\_\_5

-----------------------

(((((5\*1+3)+7)+7\*2)+2\*1\*3)+4\*2)

-----------------------

\\_\_2

\\_\_\*

\\_\_4

\\_\_+

\\_\_3

\\_\_\*

\\_\_2

\\_\_+

\\_\_2

\\_\_\*

\\_\_7

\\_\_+

\\_\_7

\\_\_+

\\_\_3

\\_\_+

\\_\_5

-----------------------

(((((5+3)+7)+7\*2)+2\*3)+4\*2)

-----------------------

elena@elena-Aspire-A315-53G:~$ ./a.out

Input expression:

1\*(2+5)+2\*1\*4+1\*3

\\_\_3

\\_\_\*

\\_\_1

\\_\_+

\\_\_4

\\_\_\*

\\_\_1

\\_\_\*

\\_\_2

\\_\_+

\\_\_5

\\_\_+

\\_\_2

\\_\_\*

\\_\_1

-----------------------

((1\*(2+5)+2\*1\*4)+1\*3)

-----------------------

\\_\_3

\\_\_+

\\_\_4

\\_\_\*

\\_\_2

\\_\_+

\\_\_5

\\_\_+

\\_\_2

-----------------------

(((2+5)+2\*4)+3)

-----------------------

**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 |
| 1 | дом | 7.05.  2022 | 10:15 | В одной из функций поставила лишний символ «}». | Внимательно писать код. |  |

**10**. Замечание автора по существу работы

**11.** Выводы:

В результате выполнения работы я составила программу выполнения заданных преобразований арифметических выражений с применением деревьев, которая вводит и печатает выражения в исходном виде, преобразовывает их в деревья, выполняет заданное действие, а именно: убирает из выражений все сомножители, равные единицы, печатает результат в виде дерева и в текстовом представлении, а также приобрела необходимые для этого знания.

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

Подпись студента