Отчет по лабораторной работе № 24 по курсу ЯМП

Студент группы М8О-107Б-19 Носов Эмиль Лаймонасович, № по списку 16

Контакты www, e-mail, icq, skype_____

Работа выполнена: «27» марта 2021г.

Преподаватель: доцент каф. 806 Сластушенский Ю. В.

Входной контроль знаний с оценкой _____

Отчет сдан « » _____201 __ г., итоговая оценка _____

Подпись преподавателя _____

1. **Тема:** Деревья выражений

2. **Цель работы:** Составить программу выполнения заданных преобразований арифметических выражений с применением деревьев. Преобразование выражения реализовать в виде набора подпрограмм.
3. **Задание** (*вариант № 15*)**:** Убрать из выражения все сомножители, равные единице.
4. **Оборудование** (лабораторное):

   ЭВМ _____, процессор _____, имя узла сети _____ с ОП _____ Мб,
   НМД _____ Мб. Терминал _____ адрес _____. Принтер _____
   Другие устройства _____
   _____

   Оборудование ПЭВМ студента, если использовалось:
   Процессор: Ryzen 7 4600H с ОП 16 Гб, НМД 256 Гб. Монитор 1920x1080
   Другие устройства _____
   _____

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

   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 Windows, наименование Windows 10 версия 2004
   интерпретатор команд bash версия 5.0.17.
   Система программирования _____версия _____
   Редактор текстов _____ версия _____
   Утилиты операционной системы _____
   _____
   Прикладные системы и программы _____
   _____
   Местонахождение и имена файлов программ и данных на домашнем компьютере _____
   _____

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

Простейшие случаи, в которых нечто равняется единице: возведение в нулевую степень, вычитание из некоторого числа числа меньшего на единицу, деление на то же самое, возведение единицы в некоторую степень. В программе необходимо реализовать обход дерева в центрированном порядке, для каждого умножения проверять, не являются ли его сомножители единицами, после чего при обнаружении единицы заменять умножение на оставшийся сомножитель.

7. **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>

//Parse functions
int lower(int b){
	b=(b<='z')&&(b>='a');
	return(b);
}

int upper(int b){
	b=(b<='Z') && (b>= 'A');
	return(b);
}

int digit(int b){
	b=(b<='9' && b>= '0');
	return(b);
}

int letter(int b){
	b=lower(b)||upper(b);
	return(b);
}

int obj(int b){
	b=digit(b)||letter(b);
	return(b);
}

int br(int b){
	b=(b=='(')||(b==')');
	return(b);
}

//Structures
union item{
	int value;//0
	char var;//1
	char op;//2
	char br;//3
};

struct token{
	union item elem;//0
	int type;//1
```

```
	int lvl;//2
};

typedef struct tree* Tree;

struct tree {
	struct tree* left;
	struct tree* right;
	struct token value;
};

//Common uses of structures
void printoken(struct token t){
	if(t.type==0){printf("%d ",t.elem.value);}
	else if(t.type==1){printf("%c ",t.elem.var);}
	else if(t.type==2){printf("%c ",t.elem.op);}
	else if(t.type==3){printf("%c ",t.elem.br);}
}

int eq(struct token A, struct token B){

return((A.type==B.type)&&(A.elem.br==B.elem.br)&&(A.elem.value==B.elem.value)&&(A.elem.var==B.elem.var)&&(A.elem.op==B.el
em.op));
}

//Tree creation
int findZero(struct token *A, int i){
	int t1=3, t2, n=A[0].lvl, j=0;
	while(i!=0){
		if(A[i].type==2){
			if((A[i].elem.op=='*')||(A[i].elem.op=='/')) t2=1;
			else if((A[i].elem.op=='+')||(A[i].elem.op=='-')) t2=0;
			else if (A[i].elem.op=='^')t2=2;
			if((A[i].lvl==n)&&(t1>t2)){t1=t2; j=i;}
			else if(A[i].lvl<n){t1=t2; n=A[i].lvl; j=i;}
			if((A[i].lvl==n)&&(t1==2)) j=i;
			else if((A[i].lvl<n)&&(t1==2)){n=A[i].lvl; j=i;}
		}
		i--;
	}
	return(j);
}

Tree ett(struct token *A, int i){
	Tree t;
	int j=findZero(A,i);
	t=(Tree)malloc(sizeof(struct tree));
	t->value=A[j];
```

```
	if(j==0){t->left=NULL; t->right=NULL;}else{
	if(j!=1) t->left=ett(A,j-1); else {t->left=(Tree)malloc(sizeof(struct tree)); t->left->value=A[j-1]; t->left->left=NULL; t->left->right=NULL;}
	if(i!=j+1) {t->right=ett(&A[j+1],i-j-1); }else {t->right=(Tree)malloc(sizeof(struct tree));  t->right->value=A[j+1]; t->right->left=NULL; t->right->right=NULL;}
	}
	return(t);
}

//Print of tree
void lRr(Tree t, int d){
	if(t!=NULL){
		lRr(t->left,d+1);
		printf("%*s",2*d,"-");
		printoken(t->value);
		printf("\n");
		lRr(t->right,d+1);
	}
}

void printtree(Tree t){
	lRr(t,0);
	printf("\n");
}

//My option
void clean(Tree *t){
	if(*t!=NULL){
		clean(&(*t)->left);
		clean(&(*t)->right);
		free(*t);
	}
}

void ChangeTo(Tree t, Tree d){
	t->value=d->value;
	if(d==t->right) clean(&t->left);
	if(d==t->left) clean(&t->right);
	t->left=d->left;
	t->right=d->right;
	free(d);
}

int isone(Tree t){
	int a=0;
	Tree tl=t->left, tr=t->right;
	if(t->value.type==2){
		//^
```

```
		if(t->value.elem.op=='^'){
			//1^_
			if((tl->value.type==0)&&(tl->value.elem.value==1)) a=1;
			//_^0
			if((tr->value.type==0)&&(tr->value.elem.value==0)) a=1;
		}
		//*
		if(t->value.elem.op=='*'){
			//1*1
			if((tl->value.type==0)&&(tr->value.type==0)&&(tl->value.elem.value==1)&&(tr->value.elem.value==1)) a=1;
		}
		///
		if(t->value.elem.op=='/'){
			//9/9
			if((tl->value.type==0)&&(tr->value.type==0)&&(tl->value.elem.value==tr->value.elem.value==1)) a=1;
		}
		//-
		if(t->value.elem.op=='-'){
			//9-8
			if((tl->value.type==0)&&(tr->value.type==0)&&(tl->value.elem.value-tr->value.elem.value==1)) a=1;
			//x-x+1
			//if() a=1;
			//x-(x-1)
			//if() a=1;
		}
		//+
		if(t->value.elem.op=='+'){
			//9+(-8)
			if((tl->value.type==0)&&(tr->value.type==2)&&(tr->value.elem.op=='*')&&(tr->left->value.type==0)&&(tr->left->value.elem.value==-1)&&(tl->value.elem.value-tr->right->value.elem.value==1)) a=1;
			//-8+9
			if((tl->value.type==2)&&(tr->value.type==0)&&(tl->value.elem.op=='*')&&(tl->left->value.type==0)&&(tl->left->value.elem.value==-1)&&(tr->value.elem.value-tl->right->value.elem.value==1)) a=1;
		}
	}
	//1
	if((t->value.type==0)&&(t->value.elem.value==1))a=1;
	return(a);
}

void findmult(Tree *t){
	if(*t!=NULL){
		findmult(&(*t)->left);
		findmult(&(*t)->right);
		if(((*t)->value.type==2)&&((*t)->value.elem.op=='*')){
			if(isone((*t)->left)){ChangeTo((*t), (*t)->right);}
			else if(isone((*t)->right)){ChangeTo((*t), (*t)->left);}
		}
```

```
        }
}

//Printing expression
int IsUM(Tree t){
        int a=0;
        if ((t->value.type==2)&&(t->value.elem.op=='*')&&(t->left->value.type==0)&&(t->left->value.elem.value==-1)) a=1;
        return a;
}

int IsBrNec(Tree t, Tree ts){
        int a=0, s;
        if(ts == t->left)s=0;
        else if(ts == t->right) s=1;
        if(t->value.type!=2) a=0;
        else if(ts->value.type!=2) a=0;
        else if(s=0){
                if((t->value.elem.op=='^')&&(ts->value.elem.op!='^')) a=1;
                else if(((t->value.elem.op=='*')||(t->value.elem.op=='/'))&&((ts->value.elem.op=='+')||(ts->value.elem.op=='-'))) a=1;
        }
        else if(s=1){
                if (IsUM(ts)) a=1;
                if((t->value.elem.op=='^')&&(ts->value.elem.op!='^')) a=1;
                else if(((t->value.elem.op=='*')||(t->value.elem.op=='/'))&&((ts->value.elem.op=='+')||(ts->value.elem.op=='-'))) a=1;
                else if((t->value.elem.op=='-')&&((ts->value.elem.op=='+')||(ts->value.elem.op=='-'))) a=1;
        }
        return a;
}

void tte (Tree t){
        if(t!=NULL){
                if(IsUM(t)){
                        printf("- ");
                        if((t->right->value.type==2)&&((t->right->value.elem.op=='+')||(t->right->value.elem.op=='-'))){
                                printf("( ");
                                tte(t->right);
                                printf(") ");
                        }
                        else tte(t->right);
                }
                else{
                        if(IsBrNec(t, t->left)){
                                printf("( ");
                                tte(t->left);
                                printf(") ");
                        }
                        else tte(t->left);
```

```
                printoken(t->value);
                if(IsBrNec(t, t->right)){
                        printf("( ");
                        tte(t->right);
                        printf(") ");
                }
                else tte(t->right);
        }
    }
}

int main(){
    struct token expr [100], tok[100];
    char a=' ', b, c=' ';
    int i=0, n=0, k=0;
    printf("Enter expression: ");
    scanf("%c", &b);
    //×òåíèå âûðàæåíèÿ
    while (b!='\n'){
        if(b=='(')n++;else if(b==')')n--;
        if(((a=='-')||(a=='+'))&&((b=='-')||(b=='+'))){printf("Too much signs"); return -3;}
        if((a=='-')&&(obj(b)||(b=='('))&&!(obj(c)||(c==')'))){expr[i-1].elem.value=-1; expr[i-1].type=0; expr[i].elem.op='*';
expr[i].type=2; i++;}
        if((a=='+')&&(obj(b)||(b=='('))&&!(obj(c))){i--;}
        if(letter(b)&&(letter(a)||digit(a))){expr[i].elem.op='*'; expr[i].type=2; i++;}
        if(digit(b)&&(letter(a)||a==')')){expr[i].elem.op='*'; expr[i].type=2; i++;}
        if((b=='(')&&(letter(a)||digit(a)||(a==')'))){expr[i].elem.op='*'; expr[i].type=2; i++;}

        if (letter(b)) {expr[i].elem.var=b; expr[i].type=1;}
        else if ((b=='(')||(b==')')){expr[i].elem.br=b; expr[i].type=3;}
        else if (b=='+' || b=='-' || b=='*' || b=='/' || b=='^') {expr[i].elem.op=b; expr[i].type=2;}
        else if(digit(b)&&!digit(a)){expr[i].elem.value=(int)b-48; expr[i].type=0;}
        else if (digit(b)&&digit(a)){i--; expr[i].elem.value*=10; expr[i].elem.value+=b-48;}
        else {printf("There is mistake in your expression"); return -1;}
        c=a;
        a=b;
        i++;
        scanf("%c", &b);
    }
    i--;
    if(n!=0){printf("Brackets error"); return -2;}
    //Ïîë÷àòü âûðàæåíèÿ
    for(int j=0; j<=i; j++) printoken(expr[j]);
    //Ðàçáèåíèå íà óðîâíè
    for(int j=0; j<=i; j++){
        if((expr[j].type<=1)||((expr[j].type==3)&&(expr[j].elem.br=='('))) expr[j].lvl=n+1;
        else if((expr[j].type==2)||((expr[j].type==3)&&(expr[j].elem.br==')'))) expr[j].lvl=n-1;
```

```
                n=expr[j].lvl;
                if (expr[j].type!=3){tok[k]=expr[j]; k++;}
        }
        k--;
        printf("\n");
        if(expr[0].lvl!=expr[i].lvl){printf("Unknown error"); return -2;}
        printf("\n");
        //Îðåîáðàçîâàíèå â äåðåâî
        Tree t=NULL;
        t=ett(tok,k);
        printf("Tree:\n");
        //Ïå÷àòü äåðåâà èñõîäíîãî âûðàæåíèÿ
        printtree(t);
        //Âûïîëíåíèå âàðèàíòà
        findmult(&t);
        //Ïå÷àòü äåðåâà ïîñëå âûïîëíåíèÿ âàðèàíòà
        printf("Result of execution of option assignment:\n");
        printtree(t);
        printf("\n\n");
        //Ïå÷àòü âûðàæåíèÿ èç äåðåâà
        tte(t);
        clean(&t);
}
```

Пункты 1-7 отчета составляются сторого до начала лабораторной работы.

Допущен к выполнению работы. **Подпись преподавателя** _____

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

```
Enter expression: 24*(1*(9-8)*5+(a-b)^0*1^(5-a))-9*(c*(8+(-7))*(-7+8)*(9-5))
24 * ( 1 * ( 9 - 8 ) * 5 + ( a - b ) ^ 0 * 1 ^ ( 5 - a ) ) - 9 * ( c * ( 8 + ( -1 * 7 ) ) * ( -1 * 7 + 8 ) * ( 9 - 5 ) )


Tree:
  -24
 -*
       -1
      -*
         -9
        --
         -8
    -*
      -5
   -+
         -a
        --
         -b
      -^
        -0
    -*
```

```
        -1
       -^
         -5
        --
         -a
--
  -9
 -*
        -c
       -*
         -8
        -+
           --1
         -*
           -7
     -*
          --1
        -*
          -7
       -+
        -8
   -*
      -9
     --
      -5
```

Result of execution of option assignment:

```
   -24
 -*
     -5
   -+
      -1
     -^
       -5
      --
       -a
--
   -9
 -*
     -c
   -*
      -9
     --
      -5
```

24 * (5 + 1 ^ (5 - a)) - 9 * c * (9 - 5)

9. **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

10. **Замечания автора** по существу работы _____

_____

_____

_____

11. **Выводы**

В ходе работы я составил программу для удаления из заданного выражения всех сомножителей, равных единице.

Недочёты при выполнении задания могут быть устранены следующим образом: _____

_____

_____

_____

_____

_____

Подпись студента _____