



# Отчет по лабораторной работе №24

по курсу: 1 фундаментальная информатика

студента группы М80-101Б-21 Тулина Ивана, № по списку: 22

Контакты www, e-mail, icq, skype: i.tulin0107@gmail.com

Работа выполнена: «17» августа 2022г.

Преподаватель: Титов В. К. каф. 806

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

Отчет сдан « \_\_\_\_\_ » 201 г., итоговая оценка \_\_\_\_\_

Подпись преподавателя \_\_\_\_\_

1.1 Тема: Деревья выражений

2 **Цель работы:** Составить программу выполнения заданных преобразований арифметических выражений с применением деревьев. Преобразование выражения в дерево рекомендуется осуществлять одним из известных методов (Рутисхаузера, Дейкстры и др.). Программа должна вводить и печатать выражения в исходном (текстовом) виде, преобразовывать их в деревья, выполнять заданные преобразования путем обращения к подпрограммам и печатать результаты в виде дерева и в текстовом представлении.

3 **Задание (вариант № 22):** Выполнить сложение и вычитание дробей:  $a/b + c/d \rightarrow (a * d + b * c) / (b * d)$ .

4 **Оборудование (лабораторное):**  
ЭВМ \_\_\_\_\_, процессор \_\_\_\_\_, имя узла сети \_\_\_\_\_ с ОП \_\_\_\_\_ Мб,  
НМД \_\_\_\_\_ Мб. Терминал \_\_\_\_\_ адрес \_\_\_\_\_. Принтер \_\_\_\_\_  
Другие устройства \_\_\_\_\_

*Оборудование ПЭВМ студента, если использовалось:*

Процессор Intel Core i5-7300HQ с ОП 7,87 Мб, НМД 15360 Мб. Монитор: встроенный  
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 UNIX, наименование Ubuntu версия 20.04.3 LTS  
интерпретатор команд bash версия \_\_\_\_\_.

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

Редактор текстов Emacs версия 3.22.30

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)

Говоря общими словами, программа должна выполнять следующие действия:

- Строить дерево выражений на основе введенного выражения
- Печатать дерево выражения
- Печатать выражение
- Преобразовывать выражение согласно заданию.

Построение дерева выражения в программе будет выполняться рекурсивно посредством *трех функций*, связанных между собой в косвенную рекурсию.

*Первая функция* выполняет роль *поиска операндов в выражении*. Она считывает операнд (число, неизвестную переменную или выражение в скобках) и конструирует терминальную вершину (для числа или переменной) или поддереву (для выражения в скобках). Работа функции состоит в считывании символа из входной строки, проверке его принадлежности к множеству цифр или латинских букв, и в случае положительного результата, создании терминальной вершины дерева. В случае отрицательного исхода проверки (функция считала открывающуюся скобку) происходит вызов функции построения выражения. Если функция считала не цифру, не букву и не скобку, она сообщит об ошибке. Результатом работы этой функции является указатель на созданный узел.

*Вторая функция* занимается *считыванием символов арифметических операций умножения и деления*. В начале своего выполнения она всегда будет вызывать функцию поиска операндов, которая вернет первый операнд операции. Затем функция считывает символ из входной строки и проверит, принадлежит ли он операциям умножения или деления. При положительном исходе проверки она создаст узел дерева выражений. Значением узла станет символ самой операции, левым поддеревом станет первый операнд, а на место указателя на правое поддерево будет помещен указатель, который вернет вызванная функция поиска операндов. Чтобы учесть случай, когда в выражении друг за другом идут сразу несколько операций умножения (деления), поместим считывание символа, проверку и создание узла в цикл **while**, который будет останавливаться, если не встретит во входной строке символ операции умножения (деления). Результатом работы этой функции является указатель на созданный узел.

*Третья функция* будет заниматься *построением дерева выражения* из указателей на поддеревья этого дерева. По сути она будет обладать тем же описанием, что и функция, считывающая символы операций умножения и деления. Различие будет в том что эта функция не считывает новых символов из входной строки. Она будет пользоваться теми символами, которые были считаны предыдущей, работавшей перед ней функцией. Другим отличием этой функции будет то, что она будет заниматься поиском символов операций сложения и вычитания (символов арифметических операций с наименьшим приоритетом выполнения). Результатом выполнения этой функции является указатель на корень созданного дерева выражений.

*Печать дерева выражений* будем производить в обратном обходе двоичного дерева. В начале каждого вызова процедуры будем увеличивать статический счетчик глубины. Затем будем вызывать процедуру для корня левого поддерева этого дерева. Потом печатать отступы, число которых равно значению счетчика глубины, и значение узла. После этого будем вызывать процедуру для корня правого поддерева этого дерева. И в конце выполнения процедуры будем уменьшать статический счетчик глубины.

*Печать выражения* будет так же происходить в процессе обратного обхода дерева. В начале каждого вызова процедуры будем проверять значение рассматриваемого узла дерева. Если в нем лежит символ сложения или вычитания, процедура должна печатать открывающуюся скобку. Затем будет происходить рекурсивный вызов для левого поддерева. Потом процедура будет выводить значение рассматриваемого узла. Затем рекурсивный вызов для правого поддерева. В конце выполнения процедуры для символов сложения и умножения будет печататься закрывающаяся скобка.

*Процедура преобразования выражения согласно заданию.*

Главной идеей этой процедуры является замена старого дерева выражений новым, в котором выполнены все необходимые преобразования. Первым делом проверим существование исходного дерева выражений. Затем проверим, принадлежит ли значение корня исходного дерева, к символам операций сложения или вычитания. Потом устроим проверку значений корней левого и правого поддерева, они должны быть равны символу деления. Только при условии что все проверки дали положительных исход процедура продолжает работу. Далее при помощи оператора **new** создаем новый узел, который будет корнем нового дерева, соответствующего дроби, полученной в результате преобразования. Заносим в значение корня нового дерева символ деления. Указателем на левое поддерево этого дерева будет указатель на числитель будущей дроби. Соответственно правый указатель – это указатель на знаменатель. В значение числителя заносим значение корня исходного дерева выражений. В значение знаменателя – символ операции умножения. В левый указатель знаменателя добавляем терминальную вершину со значением в виде значения знаменателя левого слагаемого в исходном выражении. В правый указатель – значение знаменателя правого слагаемого исходного выражения. Теперь разберемся с числителем. Левым и правым поддеревом числителя должны стать операции умножения. Слева должно находиться произведение числителя левого слагаемого и знаменателя правого слагаемого исходного выражения. Справа — наоборот — произведение числителя правого слагаемого и знаменателя левого слагаемого исходного выражения. В конце выполнения процедуры указатель на корень дерева полученной дроби должен быть записан на место указателя на корень исходного выражения.

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

```
#include<stdio.h>

struct node;
typedef node* link;
struct node
{
    char data;
    link left;
    link right;
} *tree;

char ch;
int i;

void print_tree(link tree) // Печать дерева выражений
{ static int I=0;
  I++;
  if(tree)
  { print_tree(tree->left);
    for(int i=0; i<I; i++) printf(" ");
    printf("\n__%c\n", tree->data);
    print_tree(tree->right);
  }
  I--;
}

link mknnode(char c, link l, link r) // создание узла дерева
{ link t=new node; t->data=c;
  t->left=l; t->right=r;
  return t;
}

int isAN() // проверка принадлежности к множеству цифр и латинских букв
{ return (ch>='a'&&ch<='z')||(ch>='0'&&ch<='9'); }

link expr();

link fact() // Поиск операндов в выражении
{ link t;
  scanf("%c", &ch);
  if(ch=='(')
  { t=expr(); if(ch!=')') printf("Error: not enough ')\n");
  }
  else if(isAN()) t=mknnode(ch, 0, 0);
  else {printf("Error: %c not AN\n", ch); t=0;}
  return t;
}

link term() // Поиск символов операций умножения и деления
{ link tm; char ch1;
  tm=fact(); int done=0;
  while(ch!='\n'&&!done)
  { scanf("%c", &ch);
    if(ch=='*'||ch=='/')
    { ch1=ch; tm=mknnode(ch1, tm, fact());}
    else done=1;
  }
  return tm;
}

link expr() // Поиск символов операций сложения и умножения
{ link ex; char ch1;
  ex=term(); int done=0;
  while(ch!='\n'&&!done)
  { if(ch=='+'||ch=='-')
    { ch1=ch; ex=mknnode(ch1, ex, term()); }
    else done=1;
  }
  return ex;
}
```

```

void tree_to_expr(link t) // Печать выражения
{ if(t)
  { if(t->data=='+'||t->data=='-') printf("(");
    tree_to_expr(t->left);
    printf("%c", t->data);
    tree_to_expr(t->right);
    if(t->data=='+'||t->data=='-') printf(")");
  }
}

int isFracExpr(link t)
{ return (t->data=='+'||t->data=='-')&&(t->left->data=='/'&&t->right->data=='/'); }

void action(link &t)
{ if(t)
  { if(isFracExpr(t))
    { link nt=new node; nt->data='/';
      { link frac=new node; frac->data='*';
        frac->left=t->left->right;
        frac->right=t->right->right;
        nt->right=frac;
      }
      { link num=new node; num->data=t->data;
        num->left=new node; num->left->data='*';
        num->left->left=t->left->left;
        num->left->right=t->right->right;
        num->right=new node; num->right->data='*';
        num->right->left=t->right->left;
        num->right->right=t->left->right;
        nt->left=num;
      }
      t=nt; i=1;
    }
    else
    { action(t->left);
      action(t->right);
    }
  }
}

void border()
{ printf("\n=====\\n");}

void menu()
{ border();
  printf("\n__Menu__\\n"
    "0. Exit\\n"
    "1. Print tree expression\\n"
    "2. Print expression\\n"
    "3. Trans expression\\n"
    "4. Menu\\n");
  border();
}

int main()
{ printf("Input expression:\\n");
  tree=expr();
  int k=4;
  while(k)
  { if(k==1) {border(); print_tree(tree); border();}
    else if(k==2) {border();tree_to_expr(tree);border();}
    else if(k==3)
    { i=1; while(i){i=0; action(tree); }
      printf("\\n_DONE_\\n");
    }
    else if(k==4) menu();
    else printf("Error: command not found");
    scanf("%d", &k);
  }
  return 0;}

```

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

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

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

yusayu@YS:~/Рабочий стол/cppProjects\$ cat head

```
*****
*          Лабораторная работа №24          *
*          Дерево выражений                  *
*          Выполнил: Тулин Иван Денисович    *
*          (номер по списку: 22)              *
*          Группа: М8О-101Б-21                *
*****
```

yusayu@YS:~/Рабочий стол/cppProjects\$ cat lab24.cpp

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

```
struct node;
typedef node* link;
struct node
{
    char data;
    link left;
    link right;
} *tree;
```

```
char ch;
int i;
```

```
void print_tree(link tree) // Печать дерева выражений
{ static int I=0;
  I++;
  if(tree)
  { print_tree(tree->left);
    for(int i=0; i<I; i++) printf("  ");
    printf("\_\_%c\n", tree->data);
    print_tree(tree->right);
  }
  I--;
}
```

```
link mknode(char c, link l, link r) // создание узла дерева
{ link t=new node; t->data=c;
  t->left=l; t->right=r;
  return t;
}
```

```
int isAN() // проверка принадлежности к множеству цифр и латинских букв
{ return (ch>='a'&&ch<='z')||(ch>='0'&&ch<='9'); }
```

```
link expr();
```

```
link fact() // Поиск операндов в выражении
{ link t;
  scanf("%c", &ch);
  if(ch=='(')
  { t=expr(); if(ch!=')') printf("Error: not enough ')\n");
  }
  else if(isAN()) t=mknode(ch, 0, 0);
  else {printf("Error: %c not AN\n", ch); t=0;}
  return t;
}
```

```
link term() // Поиск символов операций умножения и деления
{ link tm; char ch1;
  tm=fact(); int 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() // Поиск символов операций сложения и умножения
{ link ex; char ch1;
  ex=term(); int done=0;
  while(ch!='\n'&&!done)
  { if(ch=='+'||ch=='-')
    { ch1=ch; ex=mknnode(ch1, ex, term()); }
    else done=1;
  }
  return ex;
}

void tree_to_expr(link t) // Печать выражения
{ if(t)
  { if(t->data=='+'||t->data=='-') printf("(");
    tree_to_expr(t->left);
    printf("%c", t->data);
    tree_to_expr(t->right);
    if(t->data=='+'||t->data=='-') printf(")");
  }
}

int isFracExpr(link t)
{ return (t->data=='+'||t->data=='-')&&(t->left->data=='/'&&t->right->data=='/'); }

void action(link &t)
{ if(t)
  { if(isFracExpr(t))
    { link nt=new node; nt->data='/';
      { link frac=new node; frac->data='*';
        frac->left=t->left->right;
        frac->right=t->right->right;
        nt->right=frac;
      }
      { link num=new node; num->data=t->data;
        num->left=new node; num->left->data='*';
        num->left->left=t->left->left;
        num->left->right=t->right->right;
        num->right=new node; num->right->data='*';
        num->right->left=t->right->left;
        num->right->right=t->left->right;
        nt->left=num;
      }
      t=nt; i=1;
    }
    else
    { action(t->left);
      action(t->right);
    }
  }
}

void border()
{ printf("\n===== \n");}

void menu()
{ border();
  printf("\n__Menu__\n"
    "0. Exit\n"
    "1. Print tree expression\n"
    "2. Print expression\n"
    "3. Trans expression\n"
    "4. Menu\n");
  border();
}

int main()
{ printf("Input expression:\n");
  tree=expr();
  int k=4;
}

```

$$\begin{array}{c} \sqrt{\quad}^2 \\ \sqrt{\quad}^* \\ \sqrt{\quad}^2 \\ \sqrt{\quad}^- \\ \sqrt{\quad}^q \\ \sqrt{\quad}^- \\ \sqrt{\quad}^4 \\ \sqrt{\quad}^* \\ \sqrt{\quad}^p \\ \sqrt{\quad}^* \\ \sqrt{\quad}^n \\ \sqrt{\quad}^+ \end{array}$$

1



=====

=====

=====

=====

=====

=====

=====

\_DONE\_

=====

=====

0

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 выполнения работы я научился составлять программы, работающие с деревьями выражений, на языке Си.

Недочёты при выполнении задания могут быть устранены следующим образом: \_\_\_\_\_

Подпись студента \_\_\_\_\_