Отчет по лабораторной работе №23

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: «10» августа 2022г.

Преподаватель: Титов В. К. каф. 806

Входной контроль знаний с оценкой _____

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

Подпись преподавателя _____

1.1 **Тема:** Динамические структуры данных. Обработка деревьев.

2 **Цель работы:** Составить программу на языке Си для построения и обработки дерева общего вида, содержащего узлы типа float. Основные функции работы с деревьями реализовать в виде универсальных процедур или функций. После того как дерево создано, его обработка должна производиться в режиме текстового меню со следующими действиями: добавление нового узла, текстовая визуализация дерева, удаление узла, вычисление значения некоторой функции от дерева.

3 **Задание** (*вариант №* 30)**:** Функция - определить число нетерминальных вершин дерева (общего вида).

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)

Обработка деревьев представляет из себя работу с узлами и ссылками на них. Для упрощения обработки будем представлять общее дерево в двоичном виде. Структура узла такого дерева состоит из ссылки на сына, ссылки на брата узла и значения самого узла. Большинство процедур и функций обработки деревьев реализуется рекурсивно.

В программе будет реализован следующий набор процедур (функций):

- Процедура печати дерева
- Набор функций поиска в дереве
- Набор процедур вставки в дереве
- Процедура генерации дерева со случайными узлами.
- Процедура удаления узла по значению
- Процедура уничтожения поддерева
- Функция вычисления количества нетерминальных вершин дерева

Процедура печати дерева
Визуализацию дерева будем проводить при помощи обхода в ширину, увеличивая статический счетчик (***static int I***) с каждым рекурсивным вхождением в процедуру. Этот счетчик нужен для отсчитывания отступов при печати очередного узла, которые будут выводиться на печать в цикле ***for***.

Набор функций поиска в дереве включает в себя *процедуру поиска узла по значению* узла, *функцию поиска отца узла по значению* этого узла и *функцию поиска старшего брата узла* по его значению.

Поиск узла по его значению
Реализуем путем обхода в глубину. Если значение узла равно заданному ключу, функция возвращает ссылку на этот узел. В противном случае происходит рекурсивный вызов функции для сына этого узла. Если проверка символа возвращает ноль (проверка поддеревьев узла не дала результатов), функция вызывается для младшего брата узла.

Поиск отца узла по значению
Снова применим обход в глубину. Если значение сына узла равно заданному ключу, функция возвращает ссылку на этот узел. В противном случае локальной переменной присваивается ссылка на сына узла (старшего сына), и затем в цикле проверяются все его младшие братья (иными словами младшие сыновья этого узла). Если искомый узел так и не найден, происходит вызов функции сначала для сына, затем для младшего брата.

Поиск старшего брата узла по значению узла
Применяется обход в ширину. Сначала проверим значение младшего брата узла. Если оно равно ключу, возвращаем значение функции, в противном случае вызываем функцию сначала для младшего брата узла, затем для его сына.

Набор процедур вставки включает процедуру *вставки младшего сына узла* и *вставки младшего брата узла.* Однако вставка в этих процедурах будет производиться применительно конкретно к тому узлу, ссылка на который передается в качестве фактического аргумента при вызове функции. Для того чтобы добавлять сыновей и братьев узлам исходного дерева, имеющим конкретное значение, будем комбинировать эти процедуры с функцией поиска узла. Также в набор входит функция добавления *терминальной вершины случайному поддереву*.

Вставка младшего брата
Если при вызове процедуры в параметры ей передается пустой (нулевой) указатель, то процедура создает новый узел при помощи оператора **new**, присваивая ссылку переменной-указателю, и записывает в него передаваемое в параметре значение. А ссылки на сына и брата инициализируются нулями. Если переданный указатель отличен от нуля, то происходит вызов процедуры для младшего брата узла.

Вставка (младшего) сына узла
Если указатель-сын переданного указателя пуст, то, как и в предыдущей процедуре, создается новый, инициализированный заданным значением и содержащем нули в полях-ссылках на брата и сына. В противном случае происходит вызов функции вставки младшего брата для сына переданного узла.

Вставка терминальной вершины на случайное место в поддереве
При пустом переданном указателе, как и ранее, создает новый узел и заполняет его поля соответствующими значениями. Если указатель не пустой, случайным образом (при помощи ***rand()%2***) вызывается эта же функция для младшего брата или сына узла.

Дополнительно в программе реализуем возможность *добавления корня исходному дереву*. Для этого достаточно создать новый узел, заполнить его необходимым значением, в поле-ссылку на брата записать ноль, а в поле-сына занести ссылку на корень изначального дерева, а затем записать в указатель на корень ссылку на созданный узел.

Процедура генерации дерева со случайными узлами
В параметры процедуры должно быть помещено число вершин будущего дерева. В цикле процедура будет генерировать значения узлов и, вызывая функцию вставки терминальной вершины на случайное место в дереве, пополнять дерево новыми узлами. Генерировать значения узлов — случайные вещественные числа — будем путем деления случайного целого числа на случайную степень десяти.

Процедура удаления поддерева
Снова используем обход в глубину. Если узел существует, вызываем процедуру сначала для сына узла, потом для его брата, а потом, в случае если узел является терминальным, уничтожаем его оператором ***delete***.

Процедура удаления узла по значению
Первым делом найдем по значению удаляемый узел. Для сына найденного узла вызовем процедуру удаления поддерева. Проверим наличие старшего брата у найденного узла. Если старший брат есть, заменим его указатель на найденный узел, указателем на младшего брата найденного узла. Если старшего брата нет, выполним то же действие для отца найденного узла.

Функция вычисления количества нетерминальных вершин дерева
Используем обход в глубину. В начале каждого вызова функции будем проверять существование узла. Если у узла нет сына, будем увеличивать счетчик.

Для упрощения работы с программой добавим процедуру «меню», в которой кратко опишем доступный функционал.

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>
#include<time.h>

typedef float tdata;

struct node;
typedef node * link;

struct node
{
 tdata data;
 link brother;
 link son;
};

void randomize()
{
 long t; srand(time(&t));
}


void print_tree(link tree)
{ static int I=0;
 if(tree)
  {
   print_tree(tree->brother);
   for(int i=0; i<I; i++) printf("      ");
   printf("\\_%1.2f\n", tree->data);
   I++; print_tree(tree->son); I--;
  }
}


link search_node(link tree, tdata c)
{ if(tree)
 { if(tree->data==c) return tree;
   link t=search_node(tree->son, c);
   if(!t)
    t=search_node(tree->brother, c);
   return t;
 } return 0;
}


link search_father(link tree, tdata c)
{ if(tree)
 { if(tree->son&&tree->son->data==c)
         return tree;
   link t=tree->son;
   while(t)
   { if(t->brother&&t->brother->data==c)
            return tree;
            t=t->brother;
   }
   t=0; t=search_father(tree->son, c);
   if(!t) t=search_father(tree->brother, c);
   return t;
 } return 0;
}


link search_brother(link tree, tdata c)
{ if(tree)
 { if(tree->brother&&tree->brother->data==c)
             return tree;
   link t=search_brother(tree->brother, c);
   if(!t) t=search_brother(tree->son, c);
   return t;
 } return 0;
}
```

```
void insert_node(link &tree, tdata c)
{ if(!tree){
           tree=new node; tree->data=c;
           tree->son=0;tree->brother=0;
          }
      else
          {
           int r=rand()%2;
           if(r)
              insert_node(tree->son, c);
           else
           insert_node(tree->brother, c);
          }
}


void insert_brother(link &tree, tdata c)
{ if(!tree){
           tree=new node; tree->data=c;
           tree->son=0; tree->brother=0;
          }
    else
  insert_brother(tree->brother, c);
}


void insert_son(link &tree, tdata c)
{ if(!(tree->son)){
           tree->son=new node; tree->son->data=c;
           tree->son->son=0; tree->son->brother=0;
          }
    else
    insert_brother(tree->son, c);
}


void gen_tree(link &tree, int n)
{ int dec=1, p=rand()%2+1;
  for(int j=0; j<p; j++)  dec*=10;
  tdata val=rand()%100/((float) dec);
  if(!tree)insert_node(tree, val);
  else n++;
  for(int i=1; i<n; i++)
  { dec=1; p=rand()%2+1;
    for(int j=0; j<p; j++) dec*=10;
    val=rand()%100/((float) dec);
    insert_node(tree->son, val);
  }
}


void delete_tree(link &tree)
{ if(tree)
  { delete_tree(tree->son);
    delete_tree(tree->brother);
    if(tree->son==0&&tree->brother==0)
    { delete tree; return; }
  }
}


void delete_node(link &tree, tdata c)
{ link t=0; t=search_node(tree,c);
  if(t)
  { delete_tree(t->son);
    if(t==tree) tree=0;
    else
    { link p=0; p=search_brother(tree, t->data);
      if(!p)
      { p=search_father(tree, t->data);
        p->son=t->brother;
      }
    else
      p->brother=t->brother;
```

```
    }
  }
}


int vertex_counter(link tree)
{ if(tree)
    { int rez=0;
      if(tree->son!=0) rez++;
      rez+=vertex_counter(tree->son);
      rez+=vertex_counter(tree->brother);
      return rez;
    }
return 0;
}


void menu()
{
  printf("\n==========MENU==========="
         "\n0. Exit"
         "\n1. Genate random tree"
         "\n2. Print tree"
         "\n3. Insert son"
         "\n4. Insert brother"
         "\n5. Insert root"
         "\n6. Find father"
         "\n7. Find older brother"
         "\n8. Delete node"
         "\n9. Delete tree"
         "\n10. Action"
         "\n11. Menu\n"
         "========================\n");
}

int main(){
  randomize();
  link tree=0;
  int k=11;
  for(;;)
  { if(k==0) break;
    else if(k==1)
    { printf("\nInput number of nodes\n");
      int n; scanf("%d", &n);
      gen_tree(tree, n);
      printf("Tree was generated\n");
    }
    else if(k==2)
    { printf("==============================\n");
      print_tree(tree);
      printf("==============================\n");
    }
    else if(k==3)
    { printf("\nInput value of father\n");
      tdata c; scanf("%f", &c);
      link t=search_node(tree, c);
      if(t)
      { printf("\nInput value of node\n");
        scanf("%f", &c);
        insert_son(t, c);
      printf("Node was inserted\n");
      }
      else
        printf("Father wasn`t found\n");
    }
    else if(k==4)
    { printf("\nInput value of brother\n");
      tdata c; scanf("%f", &c);
      link t=search_node(tree, c);
      if(t)
      { printf("\nInput value of node\n");
        scanf("%f", &c);
        insert_brother(t, c);
      printf("Node was inserted\n");
      }
```

```
      else
        printf("Brother wasn`t found\n");
    }
    else if(k==5)
    { printf("\nInput value of root\n");
      tdata c; scanf("%f", &c);
      link t=tree; tree=new node;
      tree->data=c; tree->son=t;
      tree->brother=0;
      printf("Root was inserted\n");
    }
    else if(k==6)
    { printf("\nInput value of node\n");
      tdata c; scanf("%f", &c);
      link t=search_father(tree, c);
      if(t)
        printf("Father`s value is %1.2f\n", t->data);
      else
        printf("Father wasn`t found\n");
    }
    else if(k==7)
    { printf("\nInput value of node\n");
      tdata c; scanf("%f", &c);
      link t=search_brother(tree, c);
      if(t)
        printf("Older brother`s value is %1.2f\n", t->data);
      else
        printf("Older brother wasn`t found\n");
    }
    else if(k==8)
    { printf("\nInput value of node\n");
      tdata c; scanf("%f", &c);
      delete_node(tree, c);
      printf("Node was deleted\n");
    }
    else if(k==9)
    { if(tree)
        delete_node(tree, tree->data);
      printf("Tree was deleted\n");
    }
    else if(k==10)
    { int cnt=vertex_counter(tree);
      printf("\nNumber of non terminal vertex is %d", cnt);
    }
    else if(k==11) menu();
    printf("\n");
    scanf("%d", &k);
  }
printf("\n");
return 0;}
```

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).

```
yusayu@YS:~/Рабочий стол/cppProjects$ cat head
************************************************
*                  Лабораторная работа №23     *
*              Динамические структуры данных.  *
*                  Обработка деревьев          *
*              Выполнил: Тулин Иван Денисович  *
*                  (номер по списку: 22)       *
*                  Группа: М8О-101Б-21         *
************************************************
yusayu@YS:~/Рабочий стол/cppProjects$ cat lab23.cpp
#include<stdio.h>
#include<stdlib.h>
#include<time.h>

typedef float tdata;

struct node;
typedef node * link;

struct node
{
  tdata data;
  link brother;
  link son;
};

void randomize()
{
  long t; srand(time(&t));
}


void print_tree(link tree)
{ static int I=0;
  if(tree)
    {
    print_tree(tree->brother);
    for(int i=0; i<I; i++) printf("      ");
    printf("\\_%1.2f\n", tree->data);
    I++; print_tree(tree->son); I--;
    }
}


link search_node(link tree, tdata c)
{ if(tree)
  { if(tree->data==c) return tree;
    link t=search_node(tree->son, c);
    if(!t)
     t=search_node(tree->brother, c);
    return t;
  } return 0;
}


link search_father(link tree, tdata c)
{ if(tree)
  { if(tree->son&&tree->son->data==c)
          return tree;
    link t=tree->son;
    while(t)
    { if(t->brother&&t->brother->data==c)
              return tree;
              t=t->brother;
    }
    t=0; t=search_father(tree->son, c);
    if(!t) t=search_father(tree->brother, c);
    return t;
  } return 0;
}
```

```
link search_brother(link tree, tdata c)
{ if(tree)
  { if(tree->brother&&tree->brother->data==c)
              return tree;
    link t=search_brother(tree->brother, c);
    if(!t) t=search_brother(tree->son, c);
    return t;
  } return 0;
}


void insert_node(link &tree, tdata c)
{ if(!tree){
           tree=new node; tree->data=c;
           tree->son=0;tree->brother=0;
          }
      else
          {
           int r=rand()%2;
           if(r)
               insert_node(tree->son, c);
           else
           insert_node(tree->brother, c);
          }
}


void insert_brother(link &tree, tdata c)
{ if(!tree){
           tree=new node; tree->data=c;
           tree->son=0; tree->brother=0;
          }
    else
  insert_brother(tree->brother, c);
}


void insert_son(link &tree, tdata c)
{ if(!(tree->son)){
           tree->son=new node; tree->son->data=c;
           tree->son->son=0; tree->son->brother=0;
          }
    else
    insert_brother(tree->son, c);
}


void gen_tree(link &tree, int n)
{ int dec=1, p=rand()%2+1;
  for(int j=0; j<p; j++)  dec*=10;
  tdata val=rand()%100/((float) dec);
  if(!tree)insert_node(tree, val);
  else n++;
  for(int i=1; i<n; i++)
  { dec=1; p=rand()%2+1;
    for(int j=0; j<p; j++) dec*=10;
    val=rand()%100/((float) dec);
    insert_node(tree->son, val);
  }
}


void delete_tree(link &tree)
{ if(tree)
  { delete_tree(tree->son);
    delete_tree(tree->brother);
    if(tree->son==0&&tree->brother==0)
    { delete tree; return; }
  }
}


void delete_node(link &tree, tdata c)
{ link t=0; t=search_node(tree,c);
  if(t)
```

```
  { delete_tree(t->son);
    if(t==tree) tree=0;
    else
    { link p=0; p=search_brother(tree, t->data);
      if(!p)
      { p=search_father(tree, t->data);
        p->son=t->brother;
      }
    else
      p->brother=t->brother;
    }
  }
}


int vertex_counter(link tree)
{ if(tree)
    { int rez=0;
      if(tree->son!=0) rez++;
      rez+=vertex_counter(tree->son);
      rez+=vertex_counter(tree->brother);
      return rez;
    }
return 0;
}


void menu()
{
  printf("\n==========MENU=========="
         "\n0. Exit"
         "\n1. Genate random tree"
         "\n2. Print tree"
         "\n3. Insert son"
         "\n4. Insert brother"
         "\n5. Insert root"
         "\n6. Find father"
         "\n7. Find older brother"
         "\n8. Delete node"
         "\n9. Delete tree"
         "\n10. Action"
         "\n11. Menu\n"
         "========================\n");
}

int main(){
  randomize();
  link tree=0;
  int k=11;
  for(;;)
  { if(k==0) break;
    else if(k==1)
    { printf("\nInput number of nodes\n");
      int n; scanf("%d", &n);
      gen_tree(tree, n);
      printf("Tree was generated\n");
    }
    else if(k==2)
    { printf("==============================\n");
      print_tree(tree);
      printf("==============================\n");
    }
    else if(k==3)
    { printf("\nInput value of father\n");
      tdata c; scanf("%f", &c);
      link t=search_node(tree, c);
      if(t)
      { printf("\nInput value of node\n");
        scanf("%f", &c);
        insert_son(t, c);
      printf("Node was inserted\n");
      }
      else
        printf("Father wasn`t found\n");
    }
```

```
    else if(k==4)
    { printf("\nInput value of brother\n");
      tdata c; scanf("%f", &c);
      link t=search_node(tree, c);
      if(t)
      { printf("\nInput value of node\n");
        scanf("%f", &c);
        insert_brother(t, c);
      printf("Node was inserted\n");
      }
      else
        printf("Brother wasn`t found\n");
    }
    else if(k==5)
    { printf("\nInput value of root\n");
      tdata c; scanf("%f", &c);
      link t=tree; tree=new node;
      tree->data=c; tree->son=t;
      tree->brother=0;
      printf("Root was inserted\n");
    }
    else if(k==6)
    { printf("\nInput value of node\n");
      tdata c; scanf("%f", &c);
      link t=search_father(tree, c);
      if(t)
        printf("Father`s value is %1.2f\n", t->data);
      else
        printf("Father wasn`t found\n");
    }
    else if(k==7)
    { printf("\nInput value of node\n");
      tdata c; scanf("%f", &c);
      link t=search_brother(tree, c);
      if(t)
        printf("Older brother`s value is %1.2f\n", t->data);
      else
        printf("Older brother wasn`t found\n");
    }
    else if(k==8)
    { printf("\nInput value of node\n");
      tdata c; scanf("%f", &c);
      delete_node(tree, c);
      printf("Node was deleted\n");
    }
    else if(k==9)
    { if(tree)
        delete_node(tree, tree->data);
      printf("Tree was deleted\n");
    }
    else if(k==10)
    { int cnt=vertex_counter(tree);
      printf("\nNumber of non terminal vertex is %d", cnt);
    }
    else if(k==11) menu();
    printf("\n");
    scanf("%d", &k);
  }
printf("\n");
return 0;}
yusayu@YS:~/Рабочий стол/cppProjects$ g++ -o lab23.out lab23.cpp
yusayu@YS:~/Рабочий стол/cppProjects$ ./lab23.out

==========MENU==========
0. Exit
1. Genate random tree
2. Print tree
3. Insert son
4. Insert brother
5. Insert root
6. Find father
7. Find older brother
8. Delete node
9. Delete tree
10. Action
11. Menu
```

```
========================

1

Input number of nodes
15
Tree was generated

2
==============================
\_0.50
     \_3.40
          \_5.90
          \_7.30
     \_1.90
          \_2.00
               \_0.04
     \_0.93
          \_2.60
     \_0.00
          \_5.30
          \_0.83
               \_0.62
               \_9.50
                    \_0.30
==============================

3

Input value of father
0.93

Input value of node
1.46
Node was inserted

2
==============================
\_0.50
     \_3.40
          \_5.90
          \_7.30
     \_1.90
          \_2.00
               \_0.04
     \_0.93
          \_1.46
          \_2.60
     \_0.00
          \_5.30
          \_0.83
               \_0.62
               \_9.50
                    \_0.30
==============================

4

Input value of brother
9.5

Input value of node
22.1
Node was inserted

2
==============================
\_0.50
     \_3.40
          \_5.90
          \_7.30
     \_1.90
          \_2.00
               \_0.04
     \_0.93
```

```
        \_1.46
        \_2.60
    \_0.00
        \_5.30
        \_0.83
            \_22.10
            \_0.62
            \_9.50
                \_0.30
==============================

5

Input value of root
3.93
Root was inserted

2
==============================
\_3.93
    \_0.50
        \_3.40
            \_5.90
            \_7.30
        \_1.90
            \_2.00
                \_0.04
        \_0.93
            \_1.46
            \_2.60
        \_0.00
            \_5.30
            \_0.83
                \_22.10
                \_0.62
                \_9.50
                    \_0.30
==============================

11

==========MENU==========
0. Exit
1. Genate random tree
2. Print tree
3. Insert son
4. Insert brother
5. Insert root
6. Find father
7. Find older brother
8. Delete node
9. Delete tree
10. Action
11. Menu
========================

6

Input value of node
5.3
Father`s value is 0.00

7

Input value of node
1.9
Older brother`s value is 0.93

7

Input value of node
0.5
Older brother wasn`t found

6
```

```
Input value of node
4.44
Father wasn`t found

8

Input value of node
0.83
Node was deleted

2
==============================
\_3.93
    \_0.50
        \_3.40
            \_5.90
            \_7.30
        \_1.90
            \_2.00
                \_0.04
        \_0.93
            \_1.46
            \_2.60
        \_0.00
            \_5.30
==============================

9
Tree was deleted

2
==============================
==============================

1

Input number of nodes
5
Tree was generated

2
==============================
\_0.51
    \_0.42
    \_1.20
        \_5.80
            \_0.97
==============================

10

Number of non terminal vertex is 3
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 работы я научился создавать программы обработки деревьев общего вида (представленных в двоичном виде) на языке Си.

Недочёты при выполнении задания могут быть устранены следующим образом: _____

Подпись студента _____