|  |  |
| --- | --- |
|  | **Отчёт по лабораторной работе** № 23 по курсу Практикум на ЭВМ ­  студента группы М8О-108Б Жерлыгина Максима Андреевича, № по списку 8  Адреса www, e-mail, jabber, skype mmaxim2710@gmail.com  Работа выполнена: “03“ апреля 2019г.  Преподаватель: каф.806  Входной контроль знаний с оценкой  Отчёт сдан “ “ 20 г., итоговая оценка  Подпись преподавателя |

1. **Тема**: Динамическая структура данных. Обработка деревьев.
2. **Цель работы**: Получить навыки обработки деревьев.
3. **Задание** (*вариант №* 8 ): Определить количество вершин дерева, степень которых совпадает со значением элемента
4. **Оборудование** (*лабораторное*):

ЭВМ компьютер , процессор Intel Core2 Duo CPU E8500 @ 3.163GHz , имя узла сети cameron с ОП 16029 МБ

НМД 2 ГБ. Терминал gnome адрес 172.16.80.213 . Принтер Лазерный с технологией pulling

Другие устройства

*Оборудование ПЭВМ студента, если использовалось:*

Процессор Intel Core i5-7200U @ 4x 2.712GHz, ОП 8073 МБ, НМД 464 ГБ. Монитор

Другие устройства

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

Операционная система семейства Unix , наименование Ubuntu версия 16.04

Интерпретатор команд bash версия 4.3.48

Система программирования версия 8.0

Редактор текстов VIM 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 , наименование Ubuntu версия 18.04

Интерпретатор команд bash версия 4.4.19

Система программирования версия 8.0

Редактор текстов VIM версия

Утилиты операционной системы

Прикладные системы и программы

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.

1. Изучить обучающие материалы по теме «Деревья»

2. Написать необходимые функции для построения дерева и работе с ним.

3. Написать код, реализующий решение задачи соответсвующего варианта (8)

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

1. Написать функцию, строющую дерево.

2. Написать функцию, добавляющую потомков предкам.

3. Написать функцию, удаляющую потомков.

4. Написать функцию, распечатывающую дерево.

5. Написать функцию по сопоставлению степени узла с его значением.

*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)

mmaxim2710@DESKTOP-RDPBU3D:~/2sem/lab4$ emacs laba.c

mmaxim2710@DESKTOP-RDPBU3D:~/2sem/lab4$ cat laba.c

-#include <stdio.h>

#include <stdlib.h>

void menu(){

  printf("'0' -exit program\n");

  printf("'1' -get a node\n");

  printf("'2' -print a tree\n");

  printf("'3' -delete a node\n");

  printf("'4' -find node\n");

}

struct Item{

  float value;

  struct Item\* son;

  struct Item\* bro;

};

struct Item\* findnode(struct Item\* first,float fn){

  struct Item\* n = first;

  struct Item\* res;

  if(n){

    if(n->value==fn){return n;}

    else{

      if(n->son){

        res = findnode(n->son,fn);

        if(res->value==fn){return res;}

      }

      if(n->bro){

        res = findnode(n->bro,fn);

        if(res->value==fn){return res;}

      }

    }

  }

  return n;

}

void PT(struct Item\* first,int deep){

  struct Item\* n=first;

  for(int i=0;i<deep;i++){printf(" ");}

  if(n){

    printf("%3f\n",n->value);

    if(n->son){

      PT(n->son,deep+1);

    }

    if(n->bro){

      PT(n->bro,deep);

    }

  }

  else{printf("Empty pointer\n");}//

}

struct Item\* getroot(){

  printf("Enter the main root: ");

  struct Item\* root=malloc(sizeof(struct Item));

  if(root==0){printf("Lack of memory\n");}

  else{

    root->son=0;

    root->bro=0;

    scanf("%f",&root->value);

  }

  return root;

}

void getnode(struct Item\* first){

  struct Item\* n=malloc(sizeof(struct Item));

  if(n==0){printf("Lack of memory\n");}

  else{

    n->son=0;

    n->bro=0;

    int parent;

    printf("Enter a node: <value> <parent>\n");

    scanf("%f%d",&n->value,&parent);

    struct Item\* p;

    p = findnode(first,parent);

    if(p==first && parent!=first->value){

      printf("Parent node wasn't found\n");

    }

    else{

      if(p->son==0){p->son=n;}

      else{

        p=p->son;

        while(p->bro){p=p->bro;}

        p->bro=n;

      }

    }

  }

}

void delroot(struct Item\* first,float fn){

  struct Item\* n = first;

  struct Item\* r;

  if(n->son && n->son->value==fn){

    r = n->son;

    n->son=n->son->bro;

    r->bro=0;

    free(r);

  }

  else if(n->bro && n->bro->value==fn){

    r = n->bro;

    n->bro=n->bro->bro;

    r->bro=0;

    free(r);

  }

  else {

    if(n->son){

      delroot(n->son,fn);

    }

    if(n->bro){

      delroot(n->bro,fn);

    }

  }

}

void delete(struct Item\* r){

  if(r->son){

    delete(r->son);

    if(r->son->bro){

      delete(r->son->bro);

      free(r->son->bro);

      r->son->bro=0;

    }

    free(r->son);

    r->son=0;

  }

}

struct Item\* dsn(struct Item\* first){

  printf("Enter <value>\n");

  float fn;

  scanf("%f",&fn);

  struct Item\* n;

  n = findnode(first,fn);

  if(n->value==first->value && fn!=first->value){

    printf("Not found\n");

  }

  else{

    delete(n);

    if(first!=n){

      delroot(first,fn);

    }else{

      free(first);

      first=0;

    }

  }

  return first;

}

struct Item\* task(struct Item\* f, int deep, int\* count){

  struct Item\* n=f;

  if(n->son && n->bro){

    if(n->value == 2) \*count++;

    task(n->son,deep+1,count);

  }

  if(n->son){

    if(n->value == 1) \*count++;

    task(n->son,deep+1,count);

  }

  if(n->bro){

    if(n->value == 1) \*count++;

    task(n->bro,deep,count);

  }

  return 0;

}

int main(){

//printf("%d,%ld,%Ld",INT\_MAX,LONG\_MAX,LLONG\_MAX);

menu();

char command;

int out=0;

struct Item\* root=0;

struct Item\* node;

while(1){

  scanf("%s",&command);fflush(stdin);

  switch (command-'0'){

    case 0: out=1; break;

    case 1:

      if(root){

        getnode(root);

      }else{

        root = getroot();

      }

      menu();

    break;

    case 2:

      if(root){

        PT(root,0);

      }

      menu();

    break;

    case 3:

      if(root){

        root = dsn(root);

      }else{printf("Haven't got a tree\n");}

      menu();

    break;

    case 4:

      if(root){

        int ct=0;

        task(root,0,&ct);

        printf("The number of nodes is equal to: %f\n",ct);

      }

      else{printf("Haven't got a tree\n");}

      menu();

    break;

    default:

      printf("Such command isn't in menu\n");

    break;

  }

  if(out){break;}

}

}

mmaxim2710@DESKTOP-RDPBU3D:~/2sem/lab4$ gcc laba.c -o laba23

mmaxim2710@DESKTOP-RDPBU3D:~/2sem/lab4$ ls

bst\_operations.c code.c lab.c laba.c laba.c~ laba23 left test test.c value

[mmaxim2710@DESKTOP-RDPBU3D](mailto:mmaxim2710@DESKTOP-RDPBU3D):~/2sem/lab4$

mmaxim2710@DESKTOP-RDPBU3D:~/2sem/lab4$ ./laba23

'0' -exit program

'1' -get a node

'2' -print a tree

'3' -delete a node

'4' -find note

1

Enter the main root: 5

'0' -exit program

'1' -get a node

'2' -print a tree

'3' -delete a node

'4' -find note

1

Enter a node: <value> <parent>

2 5

'0' -exit program

'1' -get a node

'2' -print a tree

'3' -delete a node

'4' -find note

1

Enter a node: <value> <parent>

1 2

'0' -exit program

'1' -get a node

'2' -print a tree

'3' -delete a node

'4' -find note

1

Enter a node: <value> <parent>

4 2

'0' -exit program

'1' -get a node

'2' -print a tree

'3' -delete a node

'4' -find note

2

5.000000

2.000000

1.000000

4.000000

'0' -exit program

'1' -get a node

'2' -print a tree

'3' -delete a node

'4' -find note

4

The number of nodes is equal to: 1

'0' -exit program

'1' -get a node

'2' -print a tree

'3' -delete a node

'4' -find note

0

1. **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. Замечание автора по существу работы
2. Выводы Я получил навыки работы с деревьями, научился писать функции для их реализации, функции добавления узлов, удаления узлов, распечатки дерева. Получил некоторые теоретические знания о деревьях.

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

Подпись студента