|  |  |
| --- | --- |
|  | **Отчёт по лабораторной работе** №23 по курсу Языки и методы программирования­  студента группы М08-107Б-18 Хренова Геннадия Николаевича , № по списку 24  Адреса www, e-mail, jabber, skype [khrenov.gena@yandex.ru](mailto:khrenov.gena@yandex.ru)  Работа выполнена: “1“ мая 2019г.  Преподаватель: аспирант каф.806 Ридли А. Н.  Входной контроль знаний с оценкой  Отчёт сдан “ “ 2019г., итоговая оценка  Подпись преподавателя |

1. **Тема**: Динамические структуры данных. Обработка деревьев
2. **Цель работы**: Составить программу на Си для построения и обработки дерева общего или упорядоченного двоичного дерева. Основные функции реализовать в виде универсальных процедур и функций.
3. **Задание** (*вариант №* 15): Проверить, находятся ли все листья двоичного дерева на одном уровне
4. **Оборудование** (*лабораторное*):

ЭВМ Pentium , процессор 2.7 ГГц , имя узла сети с ОП 4096 МБ

НМД 2 ГБ. Терминал адрес . Принтер

Другие устройства

*Оборудование ПЭВМ студента, если использовалось:*

Процессор intel core i3 , ОП 3072 , НМД 50 ГБ. Монитор

Другие устройства

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

Операционная система семейства Unix , наименование версия

Интерпретатор команд bash версия

Система программирования версия

Редактор текстов Emacs 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 18.04LTS

Интерпретатор команд bash версия

Система программирования версия

Редактор текстов Emacs 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)

Программа состоит из следующих составляющих:

1) **bintree.h** - описание структуры дерева и обозначения всех функций

2) **bintree.c** — изложены все операции дерева, а также функция, выполняющая задание

3) **main.c** — главная часть, вызывающая функции

Для реализации необходимо создать следующие операции: создание и удаление дерева, добавление элемента, удаление элемента, печать дерева, проверка уровней листьев.

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

Для выполнения задания создаём функцию lvl которая будет считать уровень узла. Далее пробегаем по дереву и находим уровни всех листьев. Данные записываем в созданный нами вектор, причём в первой ячейке будет храниться количество найденных листьев, а затем их уровни. Далее просто сравниваем первый элемент со всеми остальными.

*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)

|  |  |
| --- | --- |
| **bintree.c**  #include "bintree.h"  Bintree tree\_create(void)  {  Bintree tree;  tree.root = NULL;  tree.size = 0;  return tree;  }  Node \*node\_create(data\_type elem)  {  Node \*new\_node;  new\_node = (Node \*)malloc(sizeof(Node));  new\_node->key = elem;  new\_node->left = NULL;  new\_node->right = NULL;  return new\_node;  }  void node\_destroy(Node \*node)  {  free(node);  node = NULL;  }  bool node\_is\_leaf(Node \*node)  {  return node->right == NULL && node->left == NULL;  }  void tree\_insert(Bintree \*tree, data\_type key)  {  Node \*tmp;  Node \*leaf;  Node \*new;  leaf = NULL;  tmp = tree->root;  new = node\_create(key);  while (tmp != NULL) {  leaf = tmp;  if (key < tmp->key)  tmp = tmp->left;  else  tmp = tmp->right;  }  if (leaf == NULL)  tree->root = new;  else {  if (key < leaf->key)  leaf->left = new;  else  leaf->right = new;  }  tree->size++;  }  Node \*search\_min\_node\_in\_subtree(Node \*subtree)  {  Node \*res;  res = subtree;  while (res->left != NULL)  res = res->left;  return res;  }  Node \*search\_max\_node\_in\_subtree(Node \*subtree)  {  Node \*res;  res = subtree;  while (res->right!= NULL)  res = res->right;  return res;  }  Node \*serch\_rec\_par\_del(Node \*node, data\_type key)  {  if (node == NULL)  return NULL;  if ((node->left != NULL && node->left->key == key) || (node->right != NULL && node->right->key == key))  return node;  if (key < node->key)  return serch\_rec\_par\_del(node->left, key);  else  return serch\_rec\_par\_del(node->right, key);  }  Node \*search\_par\_del(Bintree \*tree, data\_type key)  {  Node \*tmp;  tmp = tree->root;  return serch\_rec\_par\_del(tmp, key);  }  int del\_root(Bintree \*tree)  {  Node \*del;  Node \*min\_node;  del = tree->root;  if (tree->size == 1) {  tree->root = NULL;  node\_destroy(del);  }  else if (del->left == NULL && del->right != NULL) {  tree->root = del->right;  node\_destroy(del);  } else if (del->left != NULL && del->right == NULL) {  tree->root = del->left;  node\_destroy(del);  } else {  min\_node = search\_min\_node\_in\_subtree(del->right);  min\_node->left = del->left;  tree->root = del->right;  node\_destroy(del);  }  tree->size--;  return (1);  }  void del\_elem\_with\_one\_subtree(Node \*par, Node \*del, char c)  {  if (c == 'l') {  if (par->left == del)  par->left = del->left;  else  par->right = del->left;  } else {  if (par->left == del)  par->left = del->right;  else  par->right = del->right;  }  node\_destroy(del);  }  int tree\_del(Bintree \*tree, data\_type key)  {  Node \*par;  Node \*del;  Node \*min\_node;  if (tree->size == 0)  return -1;  if (tree->root->key == key) {  return del\_root(tree);  }  par = search\_par\_del(tree, key);  if (par == NULL)  return -1;  if (par->left != NULL && par->left->key == key)  del = par->left;  else  del = par->right;  if (node\_is\_leaf(del)) {  if (par->left != NULL && par->left == del)  par->left = NULL;  else  par->right = NULL;  node\_destroy(del);  }  else if (del->left == NULL && del->right != NULL)  del\_elem\_with\_one\_subtree(par, del, 'r');  else if (del->left != NULL && del->right == NULL)  del\_elem\_with\_one\_subtree(par, del, 'l');  else {  min\_node = search\_min\_node\_in\_subtree(del->right);  min\_node->left = del->left;  if (par->left == del)  par->left = del->right;  else  par->right = del->right;  node\_destroy(del);  }  tree->size--;  return 1;  }  void rec\_destroy(Node \*node)  {  if (node == NULL)  return ;  rec\_destroy(node->left);  rec\_destroy(node->right);  node\_destroy(node);  }  void tree\_destroy(Bintree \*tree)  {  rec\_destroy(tree->root);  tree->root = NULL;  } | **bintree.h**  #ifndef BINTREE\_H\_  #define BINTREE\_H\_  #include "stdlib.h"  #include "stdio.h"  #include "stdbool.h"  typedef int data\_type;  typedef struct elem {  data\_type key;  struct elem \*left;  struct elem \*right;  } Node;  typedef struct {  Node \*root;  int size;  } Bintree;  Bintree tree\_create(void);  Node \*node\_create(data\_type elem);  bool node\_is\_leaf(Node \*node);  void tree\_insert(Bintree \*tree, data\_type key);  int tree\_del(Bintree \*tree, data\_type key);  void tree\_destroy(Bintree \*tree);  void tree\_print(Bintree \*tree);  void leafscheck(Bintree \*tree);  #endif  **main.c**  #include "bintree.h"  #include <time.h>  void help(void)  {  printf("INSERT: Press a\n");  printf("DELETE: Press d\n");  printf("PRINT: Press p\n");  printf("CHECKLEALS: Press l\n");  printf("INSTRUCTIONs: Press i\n");  printf("EXIT: Press q\n");  }  int main(void)  {  Bintree tree;  int value;  char c;  tree = tree\_create();  help();  while (true) {  scanf("%c", &c);  switch (c) {  case 'a':  printf("Enter key: ");  scanf("%d", &value);  tree\_insert(&tree, value);  break;  case 'd':  printf("Enter key: ");  scanf("%d", &value);  if (tree\_del(&tree, value) == -1)  printf("This key is not exist\n");  break;  case 'p':  tree\_print(&tree);  break;  case 'l':    leafscheck(&tree);  break;  case 'i':  help();  break;  case 'q':  tree\_destroy(&tree);  return 0;  break;  }  }  }  **bintree.c**  void rec\_print(Node \*tmp, int h)  {  if (tmp == NULL)  return ;  rec\_print(tmp->right, h + 5);  printf("%\*d\n", h, tmp->key);  rec\_print(tmp->left, h + 5);  }  void tree\_print(Bintree \*tree)  {  rec\_print(tree->root, 0);  }  data\_type tree\_min(Bintree \*tree)  {  Node \*node;  node = search\_min\_node\_in\_subtree(tree->root);  return node->key;  }  data\_type tree\_max(Bintree \*tree)  {  Node \*node;  node = search\_max\_node\_in\_subtree(tree->root);  return node->key;  }  int firstlvl(Node \*node)  {  int q = 0;  while (!node\_is\_leaf(node)){  if(node->left != NULL){  node = node->left;  } else node = node->left;  q++;  }  return q;  }  int lvl(Node \*node, data\_type key)  {  int q = 0;  while (!node\_is\_leaf(node)){  if (key < node->key){  node = node->left;  q++;  } else{  node = node->right;  q++;  }  }  if (node\_is\_leaf(node) && node->key == key){  return q;  } else printf("error\n");  }  void leafscheck\_rec(Node \*node, Node \*mode, int\* vc)  {  if (node != NULL){  if (node\_is\_leaf(node)){  vc = (int \*)realloc(vc, (vc[0] + 1) \* sizeof(int));  vc[0]++;  vc[vc[0]] = lvl(mode, node->key);    }  leafscheck\_rec(node->left, mode, vc);  leafscheck\_rec(node->right, mode, vc);  }    }  void leafscheck (Bintree \*tree)  {  int\* vc = (int \*)malloc(sizeof(int));  vc[0] = 0;  leafscheck\_rec(tree->root, tree->root, vc);  for (int i = 1; i <= vc[0]; i++){  if (vc[1] != vc[i]){  printf("NO\n");  free(vc);  return;  }  }  printf("YES\n");  free(vc);  return;  } |

gennadii@lenovo-b560:~/workdir/23$ ./a.out

INSERT: Press a

DELETE: Press d

PRINT: Press p

CHECKLEALS: Press l

INSTRUCTIONs: Press i

EXIT: Press q

a

Enter key: 2

a

Enter key: 5

a

Enter key: 0

p

5

2

0

l

YES

a

Enter key: 1

p

5

2

1

0

l

NO

**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. Замечание автора по существу работы
2. Выводы: Бинарное дерево — это иерархическая структура данных, в которой каждый узел имеет значение (оно же является в данном случае и ключом) и ссылки на левого и правого потомка. Узел, находящийся на самом верхнем уровне (не являющийся чьим либо потомком) называется корнем. Узлы, не имеющие потомков (оба потомка которых равны NULL) называются листьями. Причина, по которой бинарные деревья используются чаще, чем n-арные деревья для поиска, состоит в том, что n-арные деревья более сложны, но обычно не обеспечивают реального преимущества скорости.

