|  |  |
| --- | --- |
|  | **Отчет по лабораторной работе № 23 по курсу**  **Фундаментальной информатики**  Студент группы М8О-103Б-23 Махмутов Дэниз Ирикович, № по списку 17  Контакты www, e-mail, icq, skype lustmodesta@gmail.com  Работа выполнена: « 25» апреля 2024 г.  Преподаватель: доцент каф. 806 Никулин С.П.  Входной контроль знаний с оценкой  Отчет сдан « » 202 г., итоговая оценка  Подпись преподавателя |

1. **Тема**: Динамические структуры данных, обработка деревьев.
2. **Цель работы**: Составить программу на языке Cи для построения и обработки дерева обшего вида или
3. упорядоченного двоичного дерева, содержащего узлы типа float int char и enum.
4. **Задание** (*вариант №*19 ): Определить ширину двоичного дерева
5. **Оборудование** (*лабораторное*):

Оборудование (лабораторное):

ЭВМ , процессор , имя узла сети c ОП Мб,

НМД Мб. Терминал адрес . Принтер

Другие устройства

Оборудование ПЭВМ студента, если использовалось:

Процессор AMD Ryzen 5 5600x, 6 Cores с ОП Ubuntu 22.04.3 LTS Мб,

НМД 486,400 Мб. Монитор

Другие устройства

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

Операционная система семейства , наименование версия

Интерпретатор команд bash версия

Система программирования gcc версия

Редактор текстов версия

Утилиты операционной системы

Прикладные системы и программы

Местонахождения и имена файлов программ и данных

*Программное обеспечение ЭВМ студента, если использовалось:*

*Операционная система семейства Linux, наименование Ubuntu версия 22.04.3 LTS*

*интерпретатор команд GNU bash версия 5.1.16(1)-release (x86/64-pc-linux-gnu)*

*Система программирования C версия 2.35*

*Редактор текстов GNU emacs версия 27.1*

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

**Структура узла (**Node**)**:

* + Узел содержит данные (data) и указатели на его левого (left) и правого (right) потомков.

**Функция создания узла (**createNode**)**:

* + Принимает значение данных и создает новый узел с этим значением.

**Функция вставки узла (**insertNode**)**:

* + Вставляет новый узел в дерево.
  + Если дерево пустое, создает новый узел.
  + Если значение нового узла меньше, чем значение текущего узла, вставляет его в левое поддерево, иначе в правое.

**Функция вывода дерева с отступами (**printTreeWithIndent**)**:

* + Рекурсивно выводит дерево с отступами для наглядности.

**Функция просмотра дерева (**viewTree**)**:

* + Выводит дерево с отступами, используя printTreeWithIndent.

**Функция вычисления ширины дерева (**treeWidth**)**:

* + Вычисляет максимальную ширину дерева путем нахождения максимальной ширины на каждом уровне.

**Функция удаления узла (**deleteNode**)**:

* + Удаляет узел с указанным значением из дерева.
  + Если у удаляемого узла нет потомков, просто удаляет его.
  + Если у удаляемого узла есть только один потомок, заменяет его на потомка.
  + Если у удаляемого узла есть два потомка, находит минимальный узел из правого поддерева, копирует его данные в удаляемый узел, а затем удаляет минимальный узел.

**Главная функция (**main**)**:

* + Создает пустое дерево.
  + Пользователь вводит команды (C, V, W, D, Q) для добавления узла, просмотра дерева, поиска ширины дерева, удаления узла и выхода из программы соответственно.

1. **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 <string.h>

typedef struct Node {

int data;

struct Node\* left;

struct Node\* right;

} Node;

Node\* createNode(int data) {

Node\* newNode = (Node\*)malloc(sizeof(Node));

newNode->data = data;

newNode->left = NULL;

newNode->right = NULL;

return newNode;

}

Node\* insertNode(Node\* root, int data) {

if (root == NULL) {

return createNode(data);

}

if (data < root->data) {

root->left = insertNode(root->left, data);

} else if (data > root->data) {

root->right = insertNode(root->right, data);

}

return root;

}

void printTreeWithIndent(Node\* node, int depth) {

if (node == NULL) {

return;

}

printTreeWithIndent(node->right, depth + 1);

for (int i = 0; i < depth; i++) {

printf(" ");

}

printf("%d\n", node->data);

printTreeWithIndent(node->left, depth + 1);

}

void viewTree(Node\* root) {

if (root == NULL) {

printf("Дерево пустое\n");

} else {

printf("Бинарное дерево с отступами:\n");

printTreeWithIndent(root, 0);

}

}

int calculateWidth(Node\* root, int level) {

if (root == NULL) {

return 0;

}

if (level == 1) {

return 1;

} else if (level > 1) {

return calculateWidth(root->left, level - 1) + calculateWidth(root->right, level - 1);

}

return 0;

}

int getHeight(Node\* root) {

if (root == NULL) {

return 0;

}

else {

int leftHeight = getHeight(root->left);

int rightHeight = getHeight(root->right);

return (leftHeight > rightHeight ? leftHeight : rightHeight) + 1;

}

}

int treeWidth(Node\* root) {

int maxWidth = 0;

int height = getHeight(root);

for (int i = 1; i <= height; i++) {

int width = calculateWidth(root, i);

if (width > maxWidth) {

maxWidth = width;

}

}

return maxWidth;

}

Node\* findMinNode(Node\* node) {

Node\* current = node;

while (current->left != NULL) {

current = current->left;

}

return current;

}

Node\* deleteNode(Node\* root, int data) {

if (root == NULL) {

return root;

}

if (data < root->data) {

root->left = deleteNode(root->left, data);

} else if (data > root->data) {

root->right = deleteNode(root->right, data);

} else {

if (root->left == NULL) {

Node\* temp = root->right;

free(root);

return temp;

} else if (root->right == NULL) {

Node\* temp = root->left;

free(root);

return temp;

}

Node\* temp = findMinNode(root->right);

root->data = temp->data;

root->right = deleteNode(root->right, temp->data);

}

return root;

}

int main() {

Node\* root = NULL;

char command;

int data;

do {

printf("Введите команду (C (CREATE) для добавления узла, V (VIEW) для просмотра дерева, W (WIDTH) для поиска ширины, D (DELETE) для удаления узла, Q (QUIT) для выхода):\n ");

scanf(" %c", &command);

switch(command) {

case 'C':

case 'c':

printf("Введите значение для нового узла: ");

scanf("%d", &data);

root = insertNode(root, data);

break;

case 'V':

case 'v':

viewTree(root);

break;

case 'W':

case 'w':

printf("Ширина бинарного дерева: %d\n", treeWidth(root));

break;

case 'D':

case 'd':

printf("Введите значение узла для удаления: ");

scanf("%d", &data);

root = deleteNode(root, data);

break;

}

} while (command != 'Q' && command != 'q');

return 0;

}

*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)

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <string.h>  typedef struct Node {  int data;  struct Node\* left;  struct Node\* right;  } Node;  Node\* createNode(int data) {  Node\* newNode = (Node\*)malloc(sizeof(Node));  newNode->data = data;  newNode->left = NULL;  newNode->right = NULL;  return newNode;  }  Node\* insertNode(Node\* root, int data) {  if (root == NULL) {  return createNode(data);  }    if (data < root->data) {  root->left = insertNode(root->left, data);  } else if (data > root->data) {  root->right = insertNode(root->right, data);  }    return root;  }  void printTreeWithIndent(Node\* node, int depth) {  if (node == NULL) {  return;  }    printTreeWithIndent(node->right, depth + 1);  for (int i = 0; i < depth; i++) {  printf(" ");  }  printf("%d\n", node->data);  printTreeWithIndent(node->left, depth + 1);  }  void viewTree(Node\* root) {  if (root == NULL) {  printf("Дерево пустое\n");  } else {  printf("Бинарное дерево с отступами:\n");  printTreeWithIndent(root, 0);  }  }  int calculateWidth(Node\* root, int level) {  if (root == NULL) {  return 0;  }    if (level == 1) {  return 1;  } else if (level > 1) {  return calculateWidth(root->left, level - 1) + calculateWidth(root->right, level - 1);  }  return 0;  }  int getHeight(Node\* root) {  if (root == NULL) {  return 0;  }  else {  int leftHeight = getHeight(root->left);  int rightHeight = getHeight(root->right);    return (leftHeight > rightHeight ? leftHeight : rightHeight) + 1;  }  }  int treeWidth(Node\* root) {  int maxWidth = 0;  int height = getHeight(root);    for (int i = 1; i <= height; i++) {  int width = calculateWidth(root, i);  if (width > maxWidth) {  maxWidth = width;  }  }    return maxWidth;  }  Node\* findMinNode(Node\* node) {  Node\* current = node;  while (current->left != NULL) {  current = current->left;  }  return current;  }  Node\* deleteNode(Node\* root, int data) {  if (root == NULL) {  return root;  }    if (data < root->data) {  root->left = deleteNode(root->left, data);  } else if (data > root->data) {  root->right = deleteNode(root->right, data);  } else {  if (root->left == NULL) {  Node\* temp = root->right;  free(root);  return temp;  } else if (root->right == NULL) {  Node\* temp = root->left;  free(root);  return temp;  }    Node\* temp = findMinNode(root->right);  root->data = temp->data;  root->right = deleteNode(root->right, temp->data);  }    return root;  }  int main() {  Node\* root = NULL;  char command;  int data;    do {  printf("Введите команду (C (CREATE) для добавления узла, V (VIEW) для просмотра дерева, W (WIDTH) для поиска ширины, D (DELETE) для удаления узла, Q (QUIT) для выхода):\n ");  scanf(" %c", &command);  switch(command) {    case 'C':  case 'c':  printf("Введите значение для нового узла: ");  scanf("%d", &data);  root = insertNode(root, data);  break;  case 'V':  case 'v':  viewTree(root);  break;  case 'W':  case 'w':  printf("Ширина бинарного дерева: %d\n", treeWidth(root));  break;  case 'D':  case 'd':  printf("Введите значение узла для удаления: ");  scanf("%d", &data);  root = deleteNode(root, data);  break;  }  } while (command != 'Q' && command != 'q');    return 0;  }  unix@DESKTOP-MPQDBS2:~/labs$ gcc c3.c  unix@DESKTOP-MPQDBS2:~/labs$ ./a.out  Введите команду (C (CREATE) для добавления узла, V (VIEW) для просмотра дерева, W (WIDTH) для поиска ширины, D (DELETE) для удаления узла, Q (QUIT) для выхода):  C  Введите значение для нового узла: 10  Введите команду (C (CREATE) для добавления узла, V (VIEW) для просмотра дерева, W (WIDTH) для поиска ширины, D (DELETE) для удаления узла, Q (QUIT) для выхода):  C  Введите значение для нового узла: 9  Введите команду (C (CREATE) для добавления узла, V (VIEW) для просмотра дерева, W (WIDTH) для поиска ширины, D (DELETE) для удаления узла, Q (QUIT) для выхода):  C  Введите значение для нового узла: 8  Введите команду (C (CREATE) для добавления узла, V (VIEW) для просмотра дерева, W (WIDTH) для поиска ширины, D (DELETE) для удаления узла, Q (QUIT) для выхода):  V  Бинарное дерево с отступами:  10  9  8  Введите команду (C (CREATE) для добавления узла, V (VIEW) для просмотра дерева, W (WIDTH) для поиска ширины, D (DELETE) для удаления узла, Q (QUIT) для выхода):  D  Введите значение узла для удаления: 9  Введите команду (C (CREATE) для добавления узла, V (VIEW) для просмотра дерева, W (WIDTH) для поиска ширины, D (DELETE) для удаления узла, Q (QUIT) для выхода):  V  Бинарное дерево с отступами:  10  8  Введите команду (C (CREATE) для добавления узла, V (VIEW) для просмотра дерева, W (WIDTH) для поиска ширины, D (DELETE) для удаления узла, Q (QUIT) для выхода):  W  Ширина бинарного дерева: 1  Введите команду (C (CREATE) для добавления узла, V (VIEW) для просмотра дерева, W (WIDTH) для поиска ширины, D (DELETE) для удаления узла, Q (QUIT) для выхода):  C  Введите значение для нового узла: 9  Введите команду (C (CREATE) для добавления узла, V (VIEW) для просмотра дерева, W (WIDTH) для поиска ширины, D (DELETE) для удаления узла, Q (QUIT) для выхода):  V  Бинарное дерево с отступами:  10  9  8  Введите команду (C (CREATE) для добавления узла, V (VIEW) для просмотра дерева, W (WIDTH) для поиска ширины, D (DELETE) для удаления узла, Q (QUIT) для выхода):  W  Ширина бинарного дерева: 1  Введите команду (C (CREATE) для добавления узла, V (VIEW) для просмотра дерева, W (WIDTH) для поиска ширины, D (DELETE) для удаления узла, Q (QUIT) для выхода):  V  Бинарное дерево с отступами:  10  9  8  Введите команду (C (CREATE) для добавления узла, V (VIEW) для просмотра дерева, W (WIDTH) для поиска ширины, D (DELETE) для удаления узла, Q (QUIT) для выхода):  C  Введите значение для нового узла: 11  Введите команду (C (CREATE) для добавления узла, V (VIEW) для просмотра дерева, W (WIDTH) для поиска ширины, D (DELETE) для удаления узла, Q (QUIT) для выхода):  C  Введите значение для нового узла: 13  Введите команду (C (CREATE) для добавления узла, V (VIEW) для просмотра дерева, W (WIDTH) для поиска ширины, D (DELETE) для удаления узла, Q (QUIT) для выхода):  C5  Введите значение для нового узла: Введите команду (C (CREATE) для добавления узла, V (VIEW) для просмотра дерева, W (WIDTH) для поиска ширины, D (DELETE) для удаления узла, Q (QUIT) для выхода):  V  Бинарное дерево с отступами:  13  11  10  9  8  5  Введите команду (C (CREATE) для добавления узла, V (VIEW) для просмотра дерева, W (WIDTH) для поиска ширины, D (DELETE) для удаления узла, Q (QUIT) для выхода):  W  Ширина бинарного дерева: 3 |

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 научились интерпретировать простые алгоритмы под С, получили и улучшили необходимые базовые навыки, необходимые для программирования на С

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

Подпись студента