**Отчёт по лабораторной работе** №\_\_23\_\_ по курсу\_\_\_1\_\_\_

студент группы М8О-107Б-18 Гамов Павел Антонович № по списку 4

Адрес (e-mail) pagamov@gmail.com

Работа выполнена: “ “ 2018г.

Преподаватель: асп. Каф. 806 Ридли А.Н.

Входной контроль знаний с оценкой

Отчёт сдан “ “ 2019 г итоговая оценка

Подпись преподавателя

1. Тема:

Динамические структуры данных. Обработка деревьев. Язык Си.

1. Цель работы:

Изучить способы создания и обработки деревьев на языке Си с использованием динамической памяти.

1. Задание: 30/31

Создать программу на языке Си для построения и обработки деревья общего или двоичного вида.

Создать корректные функции добавления, удаления узлов, удаления дерева, отображения в командную строку, а также функцию особого типа.

В моем случае это функция, возвращающая количество нетерминальных вершин дерева, то есть тех, которые имеют под собой еще вершины.

1. **Оборудование** (лабораторное):

ЭВМ Pentium , процессор2.7 ГГц , имя узла сети с ОП 4096 МБ

НМД 2 ГБ. Терминал bash адрес . Принтер

Другие устройства

Оборудование ПЭВМ студента, если использовалось:

Процессор 2.7 GHz Intel Core i5, ОП 8 GB (1867 MHz LPDDR3), НМД 256 ГБ. Монитор встроенный

Другие устройства

1. Программное обеспечение (лабораторное):

Операционная система семейства Unix , наименование Ubuntu версия 18.04.1 LTS

Интерпретатор команд bash версия 4.3.48

Система программирования версия

Редактор текстов Emacs версия 7.4

Утилиты операционной системы

Прикладные системы и программы

Местонахождения и имена файлов программ и данных

1. Программное обеспечение ЭВМ студента, если использовалось:

Операционная система семейства Unix , наименование Ubuntu версия 18.04.1 LTS

Интерпретатор команд bash версия 4.3.48

Система программирования версия

Редактор текстов GNU Emacs версия 25.2.2

Утилиты операционной системы

Прикладные системы и программы

Местонахождения и имена файлов программ и данных /home/pavel

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

Двоичное дерево — иерархическая структура данных, в которой каждый узел имеет не более двух потомков (детей). Как правило, первый называется родительским узлом, а дети называются левым и правым наследниками. Двоичное дерево не является упорядоченным ориентированным деревом.

Дерево общего вида — иерархическая структура данных, в которой каждый узел имеет не более одного потомка (сына) и не более одного брата. Нет понятия левый или правый, а есть понятия брат и сын.

Деревья могут быть применены в различных задачах. Компиляторы при своей работе создают деревья операций (Синтаксическое дерево (дерево операций) — это структура, представляющая собой результат работы синтаксического анализатора. Она отражает синтаксис конструкций входного языка и явно содержит в себе полную взаимосвязь операций.), деревья используются при некоторых алгоритмах Дейкстры по графам, для создания куч (не путать с кучей динамической распределения памяти), которые потом используются для создания очереди с приоритетом или для сортировки методом пирамиды.

Будем писать дерево общего вида, каждая вершина за и исключением корня будут иметь в себе ссылки на брата (элемент который находится на том же уровне дерева) и на сына (элемент который находится ниже в дереве).

Пункты 1-7 отчёта составляются **строго** **до** начала лабораторной работы.

Допущен к выполнению работы. Подпись преподавателя

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

Создадим меню использования и взаимодействия в main.c.

Создадим функции добавления, удаления, очищения дерева, нахождения количества нетерминальных вершин дерева.

Создадим tree.h где подключим необходимые библиотеки и внесем нужные функции.

1. **Распечатка протокола** (подклеить листинг окончательного варианта программы с текстовыми примерами, подписанный преподавателем)
2. **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 | дом | .2018 |  |  |  |  |

1. Замечание автора по существу работы
2. Выводы

Работа с деревьями является еще одним из бесконечных возможных применений возможности динамической памяти на языке Си.

Такая структура данных как дерево является неотъемлемой частью некоторых алгоритмов, часто применяется при поиске, в различных сортировках и работе с графами.

1. **Недочеты**, допущенные при выполнении задания, могут быть устранены следующим образом

Подпись студента

**makefile**

CC = gcc

LD = gcc

CCFLAGS = -Wall -pedantic -std=c99

###\_\_\_\_###

program: main.o tree.o ; $(CC) $(CCFLAGS) -g main.o tree.o -o program

main.o: main.c tree.h ; $(CC) $(CCFLAGS) -c -g main.c

tree.o: tree.c tree.h ; $(CC) $(CCFLAGS) -c -g tree.c

clean: ; rm \*.o

###\_\_\_###

**main.c**

#include <stdio.h>

#include <stdlib.h>

#include "tree.h"

char q;

int pass = 1;

int main() {

Tree \* tree = createTree();

while (pass) {

q = getchar();

//

switch (q) {

case '+':

if (addNode(&tree))

printf("no path\n");

break;

case '-':

if (delNode(&tree))

printf("no path\n");

break;

case 'p':

printTree(tree);

break;

case 'n':

printf("res: %d\n", neterTree(tree));

break;

case 'q':

deleteTree(&tree);

pass = 0;

break;

}

}

return 0;

}

**tree.h**

#ifndef TREE\_H

#define TREE\_H

#include <stdio.h>

#include <stdlib.h>

//struct

typedef struct Tree {

int value;

struct Tree \* son;

struct Tree \* brother;

} Tree;

//functions

Tree \* createTree();

void printTree(Tree \* tree);

int addNode(Tree \*\* root);

int delNode(Tree \*\* tree);

void deleteTree(Tree \*\* tree);

//

int neterTree(Tree \* tree);

#endif

**tree.c**

#include <stdio.h>

#include <stdlib.h>

#include "tree.h"

Tree \* createTree()

{

return NULL;

}

Tree \* createNode(int value)

{

Tree \* node = (Tree \*)malloc(sizeof(Tree));

node->value = value;

node->son = NULL;

node->brother = NULL;

//

return node;

}

int moveBrother(Tree \* tmp)

{

if (tmp->son->brother != NULL)

tmp->son = tmp->son->brother;

else

return 1;

return 0;

}

int moveSon(Tree \* tmp)

{

if (tmp->son->son != NULL)

tmp->son = tmp->son->son;

else

return 1;

return 0;

}

Tree \*\* findTreeVal(Tree \*\* tree, int target)

{

Tree \*\* path;

if ((\*tree)->value == target)

return tree;

if ((\*tree)->son != NULL) {

path = findTreeVal(&((\*tree)->son), target);

if (path)

return path;

}

if ((\*tree)->brother != NULL) {

path = findTreeVal(&((\*tree)->brother), target);

if (path)

return path;

}

return NULL;

}

int addNode(Tree \*\* tree)

{

int pass = 1;

char c;

//

if (\*tree == NULL) {

c = getchar();

switch (c) {

case 'r':

scanf(" %d", &pass);

\*tree = createNode(pass);

return 0;

break;

default:

printf("?\n");

return 0;

}

} else {

Tree \* tmp = createNode(0);

tmp->son = \*tree;

//

while (pass) {

c = getchar();

switch (c) {

case 'r':

if (\*tree != NULL) {

printf("?\n");

return 0;

}

break;

case 'v':

scanf(" %d ", &pass);

tmp->son = \*findTreeVal(tree, pass);

if(tmp->son == NULL)

return 1;

pass = 0;

break;

case 'b':

if (moveBrother(tmp)) {

free(tmp);

return 1;

}

break;

case 's':

if (moveSon(tmp)) {

free(tmp);

return 1;

}

break;

case ' ':

pass = 0;

break;

}

}

//

Tree \* node = createNode(0);

if (tmp->son->son == NULL) {

tmp->son->son = node;

} else {

tmp->son = tmp->son->son;

while (tmp->son->brother != NULL)

tmp->son = tmp->son->brother;

tmp->son->brother = node;

}

scanf("%d", &node->value);

free(tmp);

return 0;

}

}

void del\_node\_supp(Tree \*\* tree)

{

if ((\*tree)->son != NULL)

del\_node\_supp(&((\*tree)->son));

if ((\*tree)->brother != NULL)

del\_node\_supp(&((\*tree)->brother));

free(\*tree);

\*tree = NULL;

}

Tree \*\* pathNode(Tree \*\* tree)

{

int pass = 1;

char c;

Tree \*\* tmp = tree;

while (pass) {

c = getchar();

switch (c) {

case 'r':

return tree;

case 'v':

scanf(" %d", &pass);

return findTreeVal(tree, pass);

case 'b':

tmp = &((\*tmp)->brother);

if (\*tmp == NULL)

return NULL;

break;

case 's':

tmp = &((\*tmp)->son);

if (\*tmp == NULL)

return NULL;

break;

default:

return tmp;

}

}

return NULL;

}

int delNode(Tree \*\* tree)

{

if (\*tree == NULL) {

printf("null\n");

return 0;

} else {

Tree \*\* tmp = pathNode(tree);

if (\*tmp == NULL)

return 1;

if ((\*tmp)->son != NULL)

del\_node\_supp(&((\*tmp)->son));

if ((\*tmp)->brother != NULL) {

Tree \* pivet = \*tmp;

\*tmp = (\*tmp)->brother;

free(pivet);

} else {

if (\*tmp == \*tree)

printf("no root\n");

free(\*tmp);

\*tmp = NULL;

}

}

return 0;

}

void print\_sup(Tree \* node, int deep)

{

for (int i = 0; i < deep - 1; i++) {

putchar('|');

putchar('\t');

}

if (deep != 0)

printf("|----> ");

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

if (node->son != NULL)

print\_sup(node->son, deep + 1);

if (node->brother != NULL)

print\_sup(node->brother, deep);

}

void printTree(Tree \* tree)

{

if (tree == NULL)

printf("no tree\n");

else

print\_sup(tree, 0);

}

void deleteTree(Tree \*\* tree)

{

if (\*tree != NULL) {

if ((\*tree)->son != NULL) {

del\_node\_supp(&((\*tree)->son));

}

free(\*tree);

\*tree = NULL;

}

//printf("no root\n");

}

void countTree(Tree \* node, int \*\* count)

{

if (node->son != NULL) {

\*\*count = \*\*count + 1;

countTree(node->son, count);

}

if (node->brother != NULL)

countTree(node->brother, count);

}

int neterTree(Tree \* tree)

{

int count = 0;

int \* p = &(count);

//

if (tree->son != NULL) {

count++;

countTree(tree->son, &p);

}

if (tree->brother != NULL)

countTree(tree->brother, &p);

return count;

}

MacBook-Pro-Pavel:23laba pavelgamov$ make

gcc -Wall -pedantic -std=c99 -c -g main.c

gcc -Wall -pedantic -std=c99 -c -g tree.c

gcc -Wall -pedantic -std=c99 -g main.o tree.o -o program

MacBook-Pro-Pavel:23laba pavelgamov$ ./program

+r 1

+r 0

?

+ 0

+ 2

+ 3

+v 1 5

+v 3 100

p

1

|----> 0

|----> 2

|----> 3

| |----> 100

|----> 5

+v 0 5

+v 5 55

p

1

|----> 0

| |----> 5

| | |----> 55

|----> 2

|----> 3

| |----> 100

|----> 5

n

res: 4

-sbb

-v 5

p

1

|----> 0

|----> 2

|----> 5

-r

no root

-r

null

p

no tree

+r 10

+v 10 100

q