**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

Лабораторная работа №6

По дисциплине «Программирование»

 Деревья.

Группа: АВТ-041

Студент: Дударь М.А.

Преподаватель: Балакин В.В.

НОВОСИБИРСК 2021

1. **Постановка задачи:**

Необходимо организовать структуру данных по следующим критериям: вершина дерева содержит указатель на строку и список указателей на потомков. Размерность списка в корневой вершине - N, на каждом следующем уровне - в 2 раза больше. Написать функцию, которая при включении строки создает вершину, наиболее близкую к корню.

1. **Алгоритм программы:**
2. Считывание слова из файла и передача в контроллер. Контроллер пробует вставить слово в первые два уровня, иначе вызывает функцию вставки и заполняет дерево в ширину.
3. Вывод дерева в консоль.
4. **Принцип работы:**

Структура узлов списка и дерева.

struct Node {  
 Tnode \*child;  
 Node \*next;  
};  
struct Tnode {  
 char \*str;  
 int level;  
 Node \*head;  
};

Функция создания списка с N^(уровень) элементов.

void nCreate(Node \*\*head, int k) {  
 Node \*tmp = \*head;  
 int i;  
 for (i = 1; i <= pow(N, k); i++) {  
 tmp->child = (Tnode \*) malloc(sizeof(Tnode));  
 tmp->child = NULL;  
 tmp->next = (Node \*) malloc(sizeof(Node));  
 tmp = tmp->next;  
 }  
 tmp->next = (Node \*) malloc(sizeof(Node));  
 tmp->child = (Tnode \*) malloc(sizeof(Tnode));  
 tmp->child = NULL;  
 tmp->next = NULL;  
}

Функция проверки потомков узла дерева на возможность вставки нового значения.

Tnode \*layerCheck(Tnode \*\*thead) {  
 Tnode \*tTmp = \*thead;  
 Node nTmp = \*tTmp->head;  
 do {  
 if (tTmp->head->child == NULL) {  
 tTmp->head->child = (Tnode \*) malloc(sizeof(Tnode));  
 tTmp->head->child->level = tTmp->level + 1;  
 tTmp = tTmp->head->child;  
 tTmp->head = (Node \*) malloc(sizeof(Node));  
 nCreate(&tTmp->head, tTmp->level + 1);  
 (\*thead) = tTmp;  
 return \*thead;  
 }  
 tTmp->head = tTmp->head->next;  
 } while (tTmp->head->next != NULL);  
 tTmp->head = &nTmp;  
 (\*thead) = tTmp;  
 return NULL;  
}

Функция-контроллер вставки. Пробует вставить строку на первые два уровня дерева, иначе в цикле пытается вставить строку через функцию вставки, пошагово перебирая уровни дерева для вставки.

void addController(Tnode \*\*thead, char \*str) {  
 Tnode \*tmp = \*thead;  
 Tnode \*tSave = \*thead;  
 int flag = 0, lvl;  
 if (tmp == NULL) {  
 tmp = (Tnode \*) malloc(sizeof(Tnode));  
 tmp->str = NULL;  
 tmp->level = 0;  
 tmp->head = (Node \*) malloc(sizeof(struct Node));  
 nCreate(&tmp->head, tmp->level + 1);  
 tmp->str = (char \*) malloc(strlen(str) \* sizeof(char));  
 strcpy(tmp->str, str);  
 (\*thead) = tmp;  
 return;  
 } else {  
 Node \*nSave = tmp->head;  
 lvl = tmp->level + 1;  
 while (1) {  
 do {  
 if (insert(&tmp->head->child, str, lvl, 0)) {  
 flag = 1;  
 break;  
 }  
 tmp->head = tmp->head->next;  
 } while (tmp->head->next != NULL);  
 tmp->head = nSave;  
 if (flag)break;  
 lvl++;  
 }  
 tmp->head = nSave;  
 (\*thead) = tSave;  
 }  
}

Функция вставки строки в дерево, которая будет вызывать сама себя, пока не достигнет уровня дерева, который задал ей контроллер, и не вставит значение.

int insert(Tnode \*\*thead, char \*str, int lvl, int parentLvl) {  
 Tnode \*tSave = \*thead;  
 Tnode \*tmp = \*thead;  
 int flag = 0;  
 if (tmp == NULL) {  
 tmp = (Tnode \*) malloc(sizeof(Tnode));  
 tmp->str = NULL;  
 tmp->level = parentLvl + 1;  
 tmp->head = (Node \*) malloc(sizeof(struct Node));  
 nCreate(&tmp->head, tmp->level + 1);  
 tmp->str = (char \*) malloc(strlen(str) \* sizeof(char));  
 strcpy(tmp->str, str);  
 (\*thead) = tmp;  
 return 1;  
 }  
 Node \*nSave = tmp->head;  
 int tLvl = tmp->level;  
 if (tmp->level < lvl) {  
 if (layerCheck(&tmp) != NULL && tLvl + 1 == lvl) {  
 tmp->str = (char \*) malloc(strlen(str) \* sizeof(char));  
 strcpy(tmp->str, str);  
 tSave->head = nSave;  
 return 1;  
 } else {  
 tmp->head = nSave;  
 if (lvl > tLvl + 1)  
 do {  
 if (insert(&tmp->head->child, str, lvl, tmp->level)) {  
 flag = 1;  
 break;  
 }  
 tmp->head = tmp->head->next;  
 } while (tmp->head->next != NULL);  
 }  
 tmp->head = nSave;  
 (\*thead) = tSave;  
 if (flag) return 1;  
 return 0;  
 } else return 0;  
}

Функция-контроллер вывода дерева. Выводит строки первых два уровней дерева, далее в цикле выводит другие уровни дерева через функцию вывода, пошагово перебирая уровни дерева.

void printController(Tnode \*thead) {  
 Tnode \*tSave = thead;  
 Node \*thsave = thead->head;  
 int flag = 0, lvl;  
 if (thead == NULL) {  
 printf("Void tree\n");  
 return;  
 } else {  
 Node \*nSave = thead->head;  
 printf("%s\n", thead->str);  
 printLayer(&thead);  
 lvl = 1;  
 while (1) {  
 do {  
 if (print(thead->head->child, lvl)) {  
 flag = 1;  
 break;  
 }  
 thead->head = thsave->next;  
 thsave = thsave->next;  
 } while (thead->head->next != NULL);  
 thead->head = nSave;  
 thsave = nSave;  
 if (flag)break;  
 printf("\n");  
 lvl++;  
 }  
 thead->head = nSave;  
 (thead) = tSave;  
 }  
}

Функция вставки строки в дерево, которая будет вызывать сама себя, пока не достигнет уровня дерева, который задал ей контроллер, и не выведет значения.

int print(Tnode \*thead, int lvl) {  
 if (thead == NULL) return 1;  
 int tLvl = thead->level;  
 if (tLvl < lvl) {  
 Tnode \*tSave = thead;  
 Node \*nSave = thead->head;  
 int flag = 0;  
 thead->head = nSave;  
 if (lvl == tLvl + 1)printLayer(&thead);  
 if (lvl > tLvl + 1) {  
 do {  
 if (print(thead->head->child, lvl)) {  
 flag = 1;  
 break;  
 }  
 thead->head = thead->head->next;  
 } while (thead->head->next != NULL);  
 thead->head = nSave;  
 }  
 thead = tSave;  
 if (flag) return 1;  
 return 0;  
 } else return 0;  
}

Функция вывода значения строк потомков узла дерева.

void printLayer(Tnode \*\*thead) {  
 Tnode \*tTmp = \*thead;  
 Node \*nTmp = tTmp->head;  
 do {  
 if (tTmp->head->child == NULL) break;  
 printf("%s ", tTmp->head->child->str);  
 tTmp->head = tTmp->head->next;  
 } while (tTmp->head->next != NULL);  
 tTmp->head = nTmp;  
 (\*thead) = tTmp;  
}

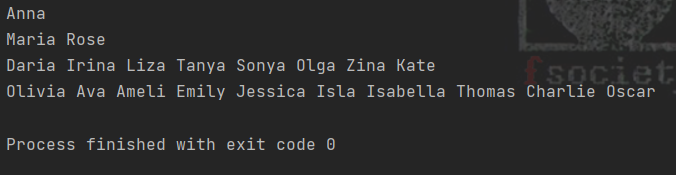
Функция считывания строк из файла и вызова их записи в дерево.

void File\_read(Tnode \*\*thead) {  
 FILE \*fp;  
 if ((fp = fopen("read.txt", "r")) == NULL) {  
 printf("Can't open file 'read.txt'\n");  
 exit(-1);  
 }  
 char str[100];  
 while (fscanf(fp, "%s", str) == 1) {  
 addController(thead, str);  
 }  
 fclose(fp);  
}

1. **Вывод:**

В ходе выполнения задания была произведена работа с деревьями и списками на языке C. Организованы добавление новых элементов в дерево и вывод дерева в консоль.

Приложение 1. Вывод программы.



Приложение 2. Текст программы.

#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <math.h>  
  
const int N = 2;  
typedef struct Tnode Tnode;  
typedef struct Node Node;  
  
struct Node {  
 Tnode \*child;  
 Node \*next;  
};  
struct Tnode {  
 char \*str;  
 int level;  
 Node \*head;  
};  
  
void nCreate(Node \*\*head, int k) {  
 Node \*tmp = \*head;  
 int i;  
 for (i = 1; i <= pow(N, k); i++) {  
 tmp->child = (Tnode \*) malloc(sizeof(Tnode));  
 tmp->child = NULL;  
 tmp->next = (Node \*) malloc(sizeof(Node));  
 tmp = tmp->next;  
 }  
 tmp->next = (Node \*) malloc(sizeof(Node));  
 tmp->child = (Tnode \*) malloc(sizeof(Tnode));  
 tmp->child = NULL;  
 tmp->next = NULL;  
}  
  
Tnode \*layerCheck(Tnode \*\*thead) {  
 Tnode \*tTmp = \*thead;  
 Node nTmp = \*tTmp->head;  
 do {  
 if (tTmp->head->child == NULL) {  
 tTmp->head->child = (Tnode \*) malloc(sizeof(Tnode));  
 tTmp->head->child->level = tTmp->level + 1;  
 tTmp = tTmp->head->child;  
 tTmp->head = (Node \*) malloc(sizeof(Node));  
 nCreate(&tTmp->head, tTmp->level + 1);  
 (\*thead) = tTmp;  
 return \*thead;  
 }  
 tTmp->head = tTmp->head->next;  
 } while (tTmp->head->next != NULL);  
 tTmp->head = &nTmp;  
 (\*thead) = tTmp;  
 return NULL;  
}  
  
int insert(Tnode \*\*thead, char \*str, int lvl, int parentLvl) {  
 Tnode \*tSave = \*thead;  
 Tnode \*tmp = \*thead;  
 int flag = 0;  
 if (tmp == NULL) {  
 tmp = (Tnode \*) malloc(sizeof(Tnode));  
 tmp->str = NULL;  
 tmp->level = parentLvl + 1;  
 tmp->head = (Node \*) malloc(sizeof(struct Node));  
 nCreate(&tmp->head, tmp->level + 1);  
 tmp->str = (char \*) malloc(strlen(str) \* sizeof(char));  
 strcpy(tmp->str, str);  
 (\*thead) = tmp;  
 return 1;  
 }  
 Node \*nSave = tmp->head;  
 int tLvl = tmp->level;  
 if (tmp->level < lvl) {  
 if (layerCheck(&tmp) != NULL && tLvl + 1 == lvl) {  
 tmp->str = (char \*) malloc(strlen(str) \* sizeof(char));  
 strcpy(tmp->str, str);  
 tSave->head = nSave;  
 return 1;  
 } else {  
 tmp->head = nSave;  
 if (lvl > tLvl + 1)  
 do {  
 if (insert(&tmp->head->child, str, lvl, tmp->level)) {  
 flag = 1;  
 break;  
 }  
 tmp->head = tmp->head->next;  
 } while (tmp->head->next != NULL);  
 }  
 tmp->head = nSave;  
 (\*thead) = tSave;  
 if (flag) return 1;  
 return 0;  
 } else return 0;  
}  
  
void addController(Tnode \*\*thead, char \*str) {  
 Tnode \*tmp = \*thead;  
 Tnode \*tSave = \*thead;  
 int flag = 0, lvl;  
 if (tmp == NULL) {  
 tmp = (Tnode \*) malloc(sizeof(Tnode));  
 tmp->str = NULL;  
 tmp->level = 0;  
 tmp->head = (Node \*) malloc(sizeof(struct Node));  
 nCreate(&tmp->head, tmp->level + 1);  
 tmp->str = (char \*) malloc(strlen(str) \* sizeof(char));  
 strcpy(tmp->str, str);  
 (\*thead) = tmp;  
 return;  
 } else {  
 Node \*nSave = tmp->head;  
 lvl = tmp->level + 1;  
 while (1) {  
 do {  
 if (insert(&tmp->head->child, str, lvl, 0)) {  
 flag = 1;  
 break;  
 }  
 tmp->head = tmp->head->next;  
 } while (tmp->head->next != NULL);  
 tmp->head = nSave;  
 if (flag)break;  
 lvl++;  
 }  
 tmp->head = nSave;  
 (\*thead) = tSave;  
 }  
}  
  
void printLayer(Tnode \*\*thead) {  
 Tnode \*tTmp = \*thead;  
 Node \*nTmp = tTmp->head;  
 do {  
 if (tTmp->head->child == NULL) break;  
 printf("%s ", tTmp->head->child->str);  
 tTmp->head = tTmp->head->next;  
 } while (tTmp->head->next != NULL);  
 tTmp->head = nTmp;  
 (\*thead) = tTmp;  
}  
  
int print(Tnode \*thead, int lvl) {  
 if (thead == NULL) return 1;  
 int tLvl = thead->level;  
 if (tLvl < lvl) {  
 Tnode \*tSave = thead;  
 Node \*nSave = thead->head;  
 int flag = 0;  
 thead->head = nSave;  
 if (lvl == tLvl + 1)printLayer(&thead);  
 if (lvl > tLvl + 1) {  
 do {  
 if (print(thead->head->child, lvl)) {  
 flag = 1;  
 break;  
 }  
 thead->head = thead->head->next;  
 } while (thead->head->next != NULL);  
 thead->head = nSave;  
 }  
 thead = tSave;  
 if (flag) return 1;  
 return 0;  
 } else return 0;  
}  
  
void printController(Tnode \*thead) {  
 Tnode \*tSave = thead;  
 Node \*thsave = thead->head;  
 int flag = 0, lvl;  
 if (thead == NULL) {  
 printf("Void tree\n");  
 return;  
 } else {  
 Node \*nSave = thead->head;  
 printf("%s\n", thead->str);  
 printLayer(&thead);  
 lvl = 1;  
 while (1) {  
 do {  
 if (print(thead->head->child, lvl)) {  
 flag = 1;  
 break;  
 }  
 thead->head = thsave->next;  
 thsave = thsave->next;  
 } while (thead->head->next != NULL);  
 thead->head = nSave;  
 thsave = nSave;  
 if (flag)break;  
 printf("\n");  
 lvl++;  
 }  
 thead->head = nSave;  
 (thead) = tSave;  
 }  
}  
  
void File\_read(Tnode \*\*thead) {  
 FILE \*fp;  
 if ((fp = fopen("read.txt", "r")) == NULL) {  
 printf("Can't open file 'read.txt'\n");  
 exit(-1);  
 }  
 char str[100];  
 while (fscanf(fp, "%s", str) == 1) {  
 addController(thead, str);  
 }  
 fclose(fp);  
}  
  
int main() {  
 Tnode \*thead = NULL;  
 File\_read(&thead);  
 printController(thead);  
 return 0;  
}