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

Лабораторная работа №7

По дисциплине «Программирование»

 Указатель на функцию.

Группа: АВТ-041

Студент: Дударь М.А.

Преподаватель: Балакин В.В.

НОВОСИБИРСК 2021

1. **Постановка задачи:**

Необходимо преобразовать функцию сортировки с использованием дерева в итератор. Проверить его работу на структуре данных, содержащих указатели на различные типы данных.

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. Реализация итератора для сортировки потомков узла дерева.
5. **Принцип работы:**

Структура узлов списка и дерева.

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

Функция создает список с количеством элементов равным количеству потомков первого узла дерева, в степени уровня слоя узла.

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);  
}

Функция-контроллер сортировки дерева, которая принимает два указателя на функции (первая функция метод сортировки, вторая алгоритм сравнивания элементов). Сортирует элементы первых два уровней дерева, далее в цикле сортирует другие уровни дерева через функцию вывода, пошагово перебирая уровни дерева.

void sortController(Tnode \*thead, void (\*sortType)(), int (\*sort)()) {  
 int flag = 0, lvl;  
 if (thead == NULL) {  
 printf("Void tree\n");  
 return;  
 } else {  
 sortType(&thead->head, sort);  
 Node \*nSave = thead->head;  
 Node \*thsave = thead->head;  
 lvl = 1;  
 while (1) {  
 do {  
 if (Sort(thead->head->child, lvl, sortType, sort)) {  
 flag = 1;  
 break;  
 }  
 thead->head = thsave->next;  
 thsave = thsave->next;  
 } while (thead->head->next != NULL);  
 thead->head = nSave;  
 thsave = nSave;  
 if (flag)break;  
 lvl++;  
 }  
 thead->head = nSave;  
 }  
}

Функция сортировки элементов дерева (принимает два указателя на функции (первая функция метод сортировки, вторая алгоритм сравнивания элементов), которая будет вызывать сама себя, пока не достигнет уровня дерева, который задал ей контроллер, и не отсортирует значения.

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

Функция сортировки списка пузырьком. Принимает указатель на функцию алгоритма сравнивания элементов.

void bubbleSort(Node \*\*head, int (\*sort)()) {  
 Node \*tmp2;  
 int flag;  
 while (1) {  
 flag = 1;  
 tmp2 = \*head;  
 while (tmp2->next->child != NULL) {  
 if (sort(tmp2->child, tmp2->next->child)) {  
 flag = 0;  
 swap(tmp2, tmp2->next, head);  
 tmp2 = getPrev(tmp2, \*head);  
 }  
 tmp2 = tmp2->next;  
 }  
 if (flag)break;  
 }  
}

Функция перестановки элементов списка.

void swap(Node \*a, Node \*b, Node \*\*head) {  
 Node \*buf = b->next;  
 if (\*head == a) {  
 b->next = a;  
 a->next = buf;  
 \*head = b;  
 return;  
 }  
 Node \*tmp = \*head;  
 Node \*prev = getPrev(a, tmp);  
 prev->next = b;  
 b->next = a;  
 a->next = buf;  
}

Функция получения предыдущего узла списка от данного.

Node \*getPrev(Node \*a, Node \*head) {  
 Node \*tmp = head;  
 if (a == head) return a;  
 while (tmp->next != a) {  
 tmp = tmp->next;  
 }  
 return tmp;  
}

Алгоритм сравнивания строк.

int strSort(Tnode \*a, Tnode \*b) {  
 int rez = strcmp(a->str, b->str);  
 if (rez == 1)return rez;  
 else return 0;  
}

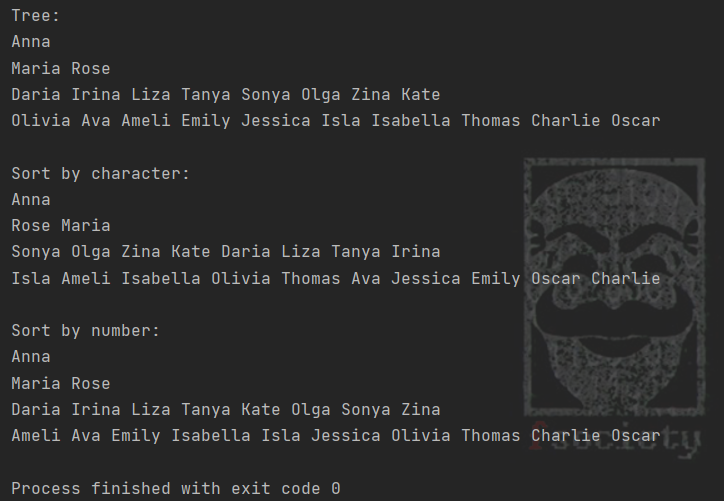
Алгоритм сравнивания целых чисел.

int intSort(Tnode \*a, Tnode \*b) {  
 if (a->value > b->value) return 1;  
 else return 0;  
}

1. **Вывод:**

В ходе выполнения задания была произведена работа с указателями на функции, деревьями и списками на языке C. Организованы сортировка дерева через итератор для по целым числам и строкам, 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 value;  
 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, int value) {  
 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);  
 tmp->value = value;  
 (\*thead) = tmp;  
 return 1;  
 }  
 int tLvl = tmp->level;  
 if (tLvl < lvl) {  
 Node \*nSave = tmp->head;  
 if (layerCheck(&tmp) != NULL && tLvl + 1 == lvl) {  
 tmp->str = (char \*) malloc(strlen(str) \* sizeof(char));  
 strcpy(tmp->str, str);  
 tmp->value = value;  
 tSave->head = nSave;  
 return 1;  
 } else {  
 tmp->head = nSave;  
 if (lvl > tLvl + 1)  
 do {  
 if (insert(&tmp->head->child, str, lvl, tmp->level, value)) {  
 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, int value) {  
 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);  
 tmp->value = value;  
 (\*thead) = tmp;  
 return;  
 } else {  
 Node \*nSave = tmp->head;  
 lvl = tmp->level + 1;  
 while (1) {  
 do {  
 if (insert(&tmp->head->child, str, lvl, 0, value)) {  
 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) {  
 Node \*nTmp = thead->head;  
 do {  
 if (thead->head->child == NULL) break;  
 printf("%s ", thead->head->child->str);  
 thead->head = thead->head->next;  
 } while (thead->head->next != NULL);  
 thead->head = nTmp;  
}  
  
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);  
 //printf("\n");  
 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];  
 int value;  
 while (fscanf(fp, "%s", str) == 1) {  
 if (!fscanf(fp, "%d", &value))exit(-1);  
 addController(thead, str, value);  
 }  
 fclose(fp);  
}  
  
int intSort(Tnode \*a, Tnode \*b) {  
 if (a->value > b->value) return 1;  
 else return 0;  
}  
  
int strSort(Tnode \*a, Tnode \*b) {  
 int rez = strcmp(a->str, b->str);  
 if (rez == 1)return rez;  
 else return 0;  
}  
  
  
Node \*getPrev(Node \*a, Node \*head) {  
 Node \*tmp = head;  
 if (a == head) return a;  
 while (tmp->next != a) {  
 tmp = tmp->next;  
 }  
 return tmp;  
}  
  
void swap(Node \*a, Node \*b, Node \*\*head) {  
 Node \*buf = b->next;  
 if (\*head == a) {  
 b->next = a;  
 a->next = buf;  
 \*head = b;  
 return;  
 }  
 Node \*tmp = \*head;  
 Node \*prev = getPrev(a, tmp);  
 prev->next = b;  
 b->next = a;  
 a->next = buf;  
}  
  
  
  
void bubbleSort(Node \*\*head, int (\*sort)()) {  
 Node \*tmp2;  
 int flag;  
 while (1) {  
 flag = 1;  
 tmp2 = \*head;  
 while (tmp2->next->child != NULL) {  
 if (sort(tmp2->child, tmp2->next->child)) {  
 flag = 0;  
 swap(tmp2, tmp2->next, head);  
 tmp2 = getPrev(tmp2, \*head);  
 }  
 tmp2 = tmp2->next;  
 }  
 if (flag)break;  
 }  
}  
  
  
int Sort(Tnode \*thead, int lvl, void (\*sortType)(), int (\*sort)()) {  
 if (thead == NULL) return 1;  
 int tLvl = thead->level;  
 if (tLvl < lvl) {  
 Node \*nSave = thead->head;  
 int flag = 0;  
 thead->head = nSave;  
 if (lvl == tLvl + 1) {  
 sortType(&thead->head, sort);  
 }  
 if (lvl > tLvl + 1) {  
 do {  
 if (Sort(thead->head->child, lvl, sortType, sort)) {  
 flag = 1;  
 break;  
 }  
 thead->head = thead->head->next;  
 } while (thead->head->next != NULL);  
 thead->head = nSave;  
 }  
 if (flag) return 1;  
 return 0;  
 } else return 0;  
}  
  
void sortController(Tnode \*thead, void (\*sortType)(), int (\*sort)()) {  
 int flag = 0, lvl;  
 if (thead == NULL) {  
 printf("Void tree\n");  
 return;  
 } else {  
 sortType(&thead->head, sort);  
 Node \*nSave = thead->head;  
 Node \*thsave = thead->head;  
 lvl = 1;  
 while (1) {  
 do {  
 if (Sort(thead->head->child, lvl, sortType, sort)) {  
 flag = 1;  
 break;  
 }  
 thead->head = thsave->next;  
 thsave = thsave->next;  
 } while (thead->head->next != NULL);  
 thead->head = nSave;  
 thsave = nSave;  
 if (flag)break;  
 lvl++;  
 }  
 thead->head = nSave;  
 }  
}  
  
int main() {  
 Tnode \*thead = NULL;  
 File\_read(&thead);  
 printf("Tree:\n");  
 printController(thead);  
 sortController(thead, bubbleSort, intSort);  
 printf("\nSort by character:\n");  
 printController(thead);  
 sortController(thead, bubbleSort, strSort);  
 printf("\nSort by number:\n");  
 printController(thead);  
 return 0;  
}