МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ**

**ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»**

**(БГТУ им. В. Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

**Лабораторная работа № 7**

по дисциплине: Алгоритмы и структуры данных

тема: «««Структуры данных типа «дерево» C»

Выполнил: ст. группы ПВ-223

Игнатьев Артур Олегович

Проверил:

асс. Солонченко Роман Евгеньевич

Белгород 2023г.

**Лабораторная работа №7**

**««Структуры данных типа «дерево» C»**

**Цель работы:** изучить СД типа «дерево», научиться их программно реализовывать и использовать.**Содержание отчета:**

1. Тема лабораторной работы.

2. Цель работы.

3. Характеристика СД типа «стек» и «очередь» (п.1 задания).

4. Индивидуальное задание.

5. Текст модуля для реализации СД типа «линейный список», текст программы для отладки модуля, тестовые данные результат работы программы.

6. Текст программы для решения задачи с использованием модуля, тестовые данные, результат работы программы.**Задание к лабораторной работе :**

1. Для СД типа «дерево» определить:

1.1. Абстрактный уровень представления СД:

1.1.1. Характер организованности и изменчивости.

1.1.2. Набор допустимых операций.

1.2. Физический уровень представления СД:

1.2.1. Схему хранения.

1.2.2. Объем памяти, занимаемый экземпляром СД.

1.2.3. Формат внутреннего представления СД и способ его интерпретации.

1.2.4. Характеристику допустимых значений.

1.2.5. Тип доступа к элементам.

1.3. Логический уровень представления СД. Способ описания СД и экземпляра СД на языке программирования.

2. Реализовать СД типа «дерево» в соответствии с вариантом индивидуального (табл.17) задания в виде модуля.

3. Разработать программу для решения задачи в соответствии с вариантом индивидуального задания (см. табл.17) с использованием модуля, полученного в результате выполнения пункта 2 задания.

Выполнение заданий:

1.1. Абстрактный уровень представления СД:

1.1.1. Характер организованности и изменчивости: динамическая, иерархическая структура данных - дерево.

1.1.2. Набор допустимых операций: инициализация, создание корня, запись данных, чтение данных, проверка — есть ли левый сын, проверка — есть ли правый сын, переход к левому сыну, переход к правому сыну, проверка — пустое ли дерево, удаление листа.

1.2. Физический уровень представления СД:

1.2.1. Схему хранения: связная схема хранения.

1.2.2. Объем памяти, занимаемый экземпляром СД: Зависит от количества элементов в дереве V = k \* size; size – размер занимаемый одним элементом структуры «дерево».

1.2.3. Формат внутреннего представления СД и способ его интерпретации: Может храниться как в статической, так и в динамической памяти.

1.2.4. Характеристику допустимых значений: CAR(БД) = ((2i)! / ((i + 1)(i!)2))·CAR(BaseType) + 1,i=1…max, где CAR(BaseType) — кардинальное число элемента БД типа BaseType, max — максимальное количество элементов в БД (не всегда определено, т.к. может зависеть от объема свободной динамической памяти).

1.2.5. Тип доступа к элементам: иерархический (от корня к листьям).

1.3. Логический уровень представления СД.

Способ описания СД и экземпляра СД на языке программирования:

typedef <BaseType> t\_base;

typedef unsigned char ptrel;

typedef struct element { t\_base data; ptrel lson, rson; } t\_element;

typedef ptrel t\_tree;

Вариант 20

2. Реализовать СД типа «дерево» в соответствии с вариантом индивидуального (табл.17) задания в виде модуля.

Файл tree.h

#ifndef ALGORITHMS\_AND\_DATA\_STRUCTURES\_TREE\_H  
#define ALGORITHMS\_AND\_DATA\_STRUCTURES\_TREE\_H  
  
#define SizeMem 100  
  
extern const short TreeOk;  
extern const short TreeNotMem;  
extern const short TreeUnder;  
  
extern short TreeError;  
  
typedef int BaseType;  
typedef unsigned char PtrEl;  
typedef struct element{  
 BaseType Data;  
 PtrEl LSon;  
 PtrEl RSon;  
}element;  
typedef PtrEl \*Tree;  
  
extern element MemTree[SizeMem];  
  
void InitTree(Tree \*T);  
void CreateRoot(Tree \*T);  
void WriteDataTree(Tree \*T, BaseType E);  
void ReadDataTree(Tree \*T, BaseType \*E);  
int IsLSon(Tree \*T);  
int IsRSon(Tree \*T);  
void MoveToLSon(Tree \*T, Tree \*TS);  
void MoveToRSon(Tree \*T, Tree \*TS);  
int IsEmptyTree(Tree \*T);  
void DellTree(Tree \*T);  
  
void InitMem();  
int EmptyMem();  
int NewMem();  
void DisposeMem(int n);  
  
#endif //ALGORITHMS\_AND\_DATA\_STRUCTURES\_TREE\_H

Файл tree.c

#include <stdio.h>  
#include <stdlib.h>  
#include "tree.h"  
  
const short TreeOk = 0;  
const short TreeNotMem = 1;  
const short TreeUnder = 2;  
  
short TreeError;  
element MemTree[SizeMem];  
  
void InitTree(Tree \*T) {  
 InitMem();  
 \*T = NULL;  
 TreeError = TreeOk;  
}  
  
void CreateRoot(Tree \*T) {  
 if (!EmptyMem()) {  
 int k = NewMem();  
 MemTree[k].LSon = 0;  
 MemTree[k].RSon = 0;  
 \*T = k;  
 TreeError = TreeOk;  
 } else {  
 TreeError = TreeNotMem;  
 }  
}  
  
void WriteDataTree(Tree \*T, BaseType E) {  
 int k = \*T;  
 MemTree[k].Data = E;  
 TreeError = TreeOk;  
}  
  
void ReadDataTree(Tree \*T, BaseType \*E) {  
 int k = \*T;  
 \*E = MemTree[k].Data;  
 TreeError = TreeOk;  
}  
  
int IsLSon(Tree \*T) {  
 int k = \*T;  
 TreeError = TreeOk;  
 return (MemTree[k].LSon != 0);  
}  
  
int IsRSon(Tree \*T) {  
 int k = \*T;  
 TreeError = TreeOk;  
 return (MemTree[k].RSon != 0);  
}  
  
void MoveToLSon(Tree \*T, Tree \*TS) {  
 int k = \*T;  
 if (IsLSon(T)) {  
 \*TS = MemTree[k].LSon;  
 TreeError = TreeOk;  
 } else {  
 TreeError = TreeUnder;  
 }  
}  
  
void MoveToRSon(Tree \*T, Tree \*TS) {  
 int k = \*T;  
 if (IsRSon(T)) {  
 \*TS = MemTree[k].RSon;  
 TreeError = TreeOk;  
 } else {  
 TreeError = TreeUnder;  
 }  
}  
  
int IsEmptyTree(Tree \*T) {  
 return !(IsLSon(T) || IsRSon(T));  
}  
  
void DellTree(Tree \*T) {  
 Tree TS;  
 if (IsRSon(T)) {  
 MoveToRSon(T, &TS);  
 DellTree(&TS);  
 }  
 if (IsLSon(T)) {  
 MoveToLSon(T, &TS);  
 DellTree(&TS);  
 }  
 int k = \*T;  
 DisposeMem(k);  
}  
  
void InitMem() {  
 for (int i = 0; i < SizeMem - 1; i++) {  
 MemTree[i].Data = NULL;  
 MemTree[i].LSon = 0;  
 MemTree[i].RSon = i + 1;  
 }  
}  
  
int EmptyMem() {  
 return (MemTree[SizeMem - 1].Data != NULL);  
}  
  
int NewMem() {  
 unsigned char t = MemTree[0].RSon;  
 MemTree[0].RSon = MemTree[t].RSon;  
 return t;  
}  
  
void DisposeMem(int n) {  
 MemTree[n].Data = NULL;  
 MemTree[n].LSon = 0;  
 MemTree[n].RSon = MemTree[0].RSon;  
 MemTree[0].RSon = n;  
}

3. Разработать программу для решения задачи в соответствии с вариантом индивидуального задания (см. табл.17) с использованием модуля, полученного в результате выполнения пункта 2 задания.

Файл main.c

#include <stdio.h>  
#include <stdlib.h>  
#include "../../libs/alg/labs/lab7/tree.h"  
  
// Глобальная переменная  
char \*s = "+ \* 2 4 \* 3 5";  
  
// Прототипы функций  
int isOperation(char c);  
int isDigit(char c);  
void BuildTree(Tree \*T);  
void WritePostfix(Tree \*T);  
int WriteCalc(Tree \*T);  
  
int main() {  
 Tree \*T;  
 InitTree(&T);  
 CreateRoot(&T);  
 printf("%s\n\n", s);  
 BuildTree(&T);  
 WritePostfix(&T);  
 puts("\n");  
 WriteCalc(&T);  
}  
  
// Функция для проверки того, является ли символ операцией  
int isOperation(char c) {  
 return (c == '/' || c == '+' || c == '-' || c == '\*');  
}  
  
// Функция для проверки, является ли символ цифрой  
int isDigit(char c) {  
 return (c >= '0' && c <= '9');  
}  
  
// Функция для построения дерева из заданного выражения  
void BuildTree(Tree \*T) {  
 Tree TS;  
 int k;  
 if (\*s != '\0') {  
 if (\*s == ' ')  
 s++;  
 if (isOperation(\*s)) {  
 WriteDataTree(T, \*s);  
 s++;  
 CreateRoot(&TS);  
 k = \*T;  
 MemTree[k].LSon = TS;  
 MoveToLSon(T, &TS);  
 BuildTree(&TS);  
 CreateRoot(&TS);  
 k = \*T;  
 MemTree[k].RSon = TS;  
 MoveToRSon(T, &TS);  
 BuildTree(&TS);  
 }  
 if (isDigit(\*s)) {  
 WriteDataTree(T, \*s - '0');  
 s++;  
 }  
 }  
}  
  
// Функция для записи постфиксного выражения  
void WritePostfix(Tree \*T) {  
 Tree TS;  
 if (IsLSon(T)) {  
 MoveToLSon(T, &TS);  
 WritePostfix(&TS);  
 }  
 if (IsRSon(T)) {  
 MoveToRSon(T, &TS);  
 WritePostfix(&TS);  
 }  
 char c;  
 ReadDataTree(T, &c);  
 if (isDigit(c + '0'))  
 c = c + '0';  
 printf("%c ", c);  
}  
  
// Функция для вычисления и записи результата выражения  
int WriteCalc(Tree \*T) {  
 Tree TS;  
 int op1, op2, res = 0;  
 char c;  
 if (IsLSon(T)) {  
 MoveToLSon(T, &TS);  
 op1 = WriteCalc(&TS);  
 }  
 if (IsRSon(T)) {  
 MoveToRSon(T, &TS);  
 op2 = WriteCalc(&TS);  
 ReadDataTree(T, &c);  
 switch (c) {  
 case '\*':  
 res = op1 \* op2;  
 break;  
 case '/':  
 res = op1 / op2;  
 break;  
 case '+':  
 res = op1 + op2;  
 break;  
 case '-':  
 res = op1 - op2;  
 break;  
 }  
 printf("%i %c %i = %i\n", op1, c, op2, res);  
 }  
 ReadDataTree(T, &c);  
 if (isOperation(c))  
 return res;  
 else  
 return c;  
}

Вывод: в ходе выполнения лабораторной работы были изучены СД типа «дерево», научиться их программно реализовывать и использовать.