**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ**

**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

дисциплина: Алгоритмы и Структуры данных

тема: «**Структуры данных типа «дерево» (Pascal/С)**»

Выполнил: ст. группы ПВ-21

Донцов Александр Алексеевич

Проверил: Синюк Василий Григорьевич

Белгород 2018

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

З а д а н и е

1. Для СД типа «дерево» определить:

1.1. Абстрактный уровень представления СД:

* + 1. Характер организованности и изменчивости.

представляет собой иерархическую структуру — дерево

Динамическая память

* + 1. 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. Физический уровень представления СД:
     1. Схему хранения.

Связная

1.2.2. Объем памяти, занимаемый экземпляром СД.

CAR(БД) = (∑(2i)! / ((i + 1)(i!)2))·CAR(BaseType) + 1,i=1…max

* + 1. Формат внутреннего представления СД и способ его интерпретации.

Располагаться БД может в статической или динамической памяти

* + 1. Характеристику допустимых значений.

CAR(БД) = (∑(2i)! / ((i + 1)(i!)2))·CAR(BaseType) + 1,i=1…max

1.2.5. Тип доступа к элементам.

1.3. Логический уровень представления СД.

Способ описания СД и экземпляра СД на языке программирования.

typedef ptrel Tree;

typedef struct element {

BaseType data;

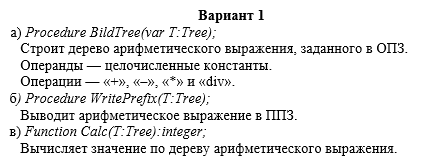
ptrel LSon;

ptrel RSon;

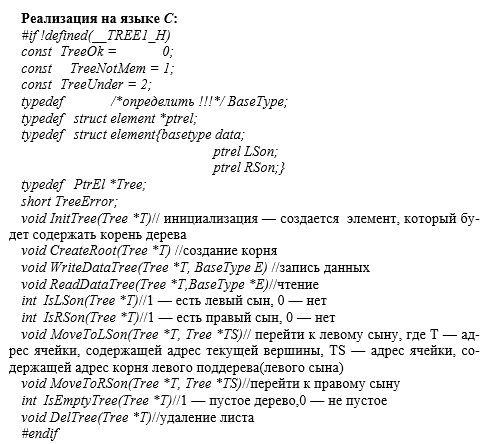
};

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.







Заголовочный модуль ListOLS.h

#pragma once

#ifndef LISTOLS\_H\_

#define LISTOLS\_H\_

// константы ошибок

const short List\_OK = 0;

const short List\_Empty = 1;

const short List\_NotMem = 2;

const short List\_End = 3;

//переменная ошибок

extern short ListError;

//базовый тип

typedef void \* BaseType;

// элемент списка

struct element {

BaseType Data; // данные

element\* Next; // указатель на следующий элемент

};

// односвязный линейный список

typedef struct {

//указатель на первый элемент списка

element\* first;

//указатель на текущий элемент

element\* current;

} List;

//конструктор

void ListInit(List \*L);

//деструктор

void ListDone(List \*L);

//добавление элемента после рабочего указателя

void ListPut(List \*L, BaseType E);

//добавление элемента в начало списка

void ListPutFirst(List \*L, BaseType E);

//взятие элемента по рабочему указателю

BaseType ListGet(List \*L);

//взятие элемента по рабочему указателю

BaseType ListRead(List \*L);

// перенос рабочего указателя на следующий элемент

void ListMoveNext(List \*L);

//являеться ли список пустым

int ListEmpty(List \*L);

//находиться ли рабочий указатель в конце списка

int ListEnd(List \*L);

//перенос указателя в конец списка

void ListPtrEnd(List \*L);

//перенос указателя в начало списка

void ListPtrBegin(List \*L);

#endif

Исходный файл ListOLS.cpp

#include "ListOLS.h"

#include "malloc.h"

short ListError;

//очищение памяти элемента

void freeElem(element \*el) { free(el); }

//конструктор

void ListInit(List \*L) {

L->first = NULL;

L->current = NULL;

ListError = List\_OK;

}

//деструктор

void ListDone(List \*L) {//

element \* del;// удаляемый элемент

element \* next; // следующий элемент

//если список не пустой

if (L->first != NULL) {

next = L->first;

do {

del = next;

freeElem(del);

next = next->Next;

} while (next != NULL);

}

L->current = NULL;

L->first = NULL;

}

//создание и инициализация элемента списка

element \*creatElem(BaseType E) {

element \* el = (element \*)malloc(sizeof(element));

//если не удалось выделить память указателю

//возвращаем нулевой указатель

if (el == NULL)

return NULL;

el->Data = E;

el->Next = NULL;

return el;

}

//добавление элемента в начало списка

void ListPutFirst(List \*L, BaseType E) {

element \*el = creatElem(E);

//если не удалось выделить память

if (el == NULL) {

//инициализируем переменную ошибки

ListError = List\_NotMem;

return;

}

//если список пуст

if (ListEmpty(L))

L->first = el;

else {

el->Next = L->first;

L->first = el;

}

ListError = List\_OK;

}

//добавление элемента после рабочего указателя

void ListPut(List \*L, BaseType E) {

//создаем элемент списка

element \*el = creatElem(E);

//если не удалось выделить память

if (el == NULL) {

//инициализируем переменную ошибки

ListError = List\_NotMem;

return;

}

//если список пуст, то вставляем в начало

if (ListEmpty(L)) {

L->first = el;

L->current = el;

}

else if (L->current->Next == NULL) {// если рабочий указатель в конце списка

L->current->Next = el;

L->current = el;

}

else {// если рабочий указатель в середине (общий случай)

el->Next = L->current->Next;

L->current->Next = el;

}

ListError = List\_OK;

}

//взятие элемента по рабочему указателю

BaseType ListRead(List \*L) {

//если список пустой, то инициализировать переменную ошибки

if (ListEmpty(L)) {

ListError = List\_Empty;

return NULL;

}

return L->current->Data;

}

//взятие(удаление) элемента по рабочему указателю

BaseType ListGet(List \*L) {

//если список пустой, то инициализировать переменную ошибки

if (ListEmpty(L)) {

ListError = List\_Empty;

return NULL;

}

//запоминаем данные текущего элемента

BaseType data = L->current->Data;

//УДАЛЯЕМ ТЕКУЩИЙ ЭЛЕМЕНТ

//1 ситуация - текущий элемент первый в списке

if (L->current == L->first) {

L->first = L->current->Next;

freeElem(L->current);

L->current = L->first;

}

else { //общий случай

element\* previus = L->first; //указатель на предыдущий элемент

while (previus->Next != L->current)

previus = previus->Next;

previus->Next = L->current->Next;

freeElem(L->current);

L->current = previus;

}

ListError = List\_OK;

return data;

}

// перенос рабочего указателя на следующий элемент

void ListMoveNext(List \*L) {

L->current = L->current->Next;

}

//являеться ли список пустым

int ListEmpty(List \*L) {

return (L->first == NULL) ? 1 : 0;

}

//находиться ли рабочий указатель в конце списка

int ListEnd(List \*L) {

return (L->current == NULL) ? 1 : 0;

}

//перенос указателя в конец списка

void ListPtrEnd(List \*L) {

//если список пустой, то инициализировать переменную ошибки

if (ListEmpty(L)) {

ListError = List\_Empty;

return;

}

L->current = L->first;

while (L->current->Next != NULL)

L->current = L->current->Next;

}

//перенос указателя в начала списка

void ListPtrBegin(List \*L) {

//если список пустой, то инициализировать переменную ошибки

if (ListEmpty(L)) {

ListError = List\_Empty;

return;

}

L->current = L->first;

}

Заголовочный модуль Stack.h

#pragma once

#ifndef STACK\_H\_

#define STACK\_H\_

#include "ListOLS.h"

const short StackOk = List\_OK;

const short StackUnder = List\_Empty;

const short StackOver = List\_NotMem;

// Переменная ошибок

extern int StackError;

typedef List Stack;

//Инициализация стека

void InitStack(Stack \*s);

//Поместить элемент в стек

void PutStack(Stack \*s, void \*E);

//Извлечь элемент из стека

void \*GetStack(Stack \*s);

//Прочитать элемент из вершины стека

void \*ReadStack(Stack \*s);

//Проверка: стек пуст?

int EmptyStack(Stack \*s);

//Уничтожить стек

void DoneStack(Stack \*s);

#endif

Исходный файл Stack.cpp

#include "Stack.h"

int StackError;

//Инициализация стека

void InitStack(Stack \*s) { ListInit(s); }

//Поместить элемент в стек

void PutStack(Stack \*s, void \*E) { ListPutFirst(s, E); }

//Извлечь элемент из стека

void \*GetStack(Stack \*s) {

//перенос рабочего указателя в начало списка

ListPtrBegin(s);

return ListGet(s);

}

//Проверка: стек пуст?

int EmptyStack(Stack \*s) { return ListEmpty(s); }

//Прочитать элемент из вершины стека

void \*ReadStack(Stack \*s) {

//перенос рабочего указателя в начало списка

ListPtrBegin(s);

return ListRead(s);

}

//Уничтожить стек

void DoneStack(Stack \*s) { ListDone(s);}

Заголовочный файл Tree.h

#pragma once

#ifndef \_\_TREE1\_H

#define \_\_TREE1\_H

const short TreeOk = 0;

const short TreeNotMem = 1;

const short TreeUnder = 2;

const short NullPoint = 3;

typedef void\* basetype;

typedef struct elementT \*ptrel;

typedef struct elementT {

basetype data;

ptrel LSon;

ptrel RSon;

};

typedef ptrel Tree;

extern short TreeError;

void InitTree(Tree \*T);// инициализация — создается элемент, который бу-дет содержать корень дерева

void CreateRoot(Tree \*T); //создание корня

void WriteDataTree(Tree \*T, basetype E); //запись данных

basetype ReadDataTree(Tree \*T);//чтение

int IsLSon(Tree \*T);//1 — есть левый сын, 0 — нет

int IsRSon(Tree \*T);//1 — есть правый сын, 0 — нет

void MoveToLSon(Tree \*T, Tree \*TS);// перейти к левому сыну, где T — ад-рес ячейки, содержащей адрес текущей вершины, TS — адрес ячейки, со-держащей адрес корня левого поддерева(левого сына)

void MoveToRSon(Tree \*T, Tree \*TS);//перейти к правому сыну

int IsEmptyTree(Tree \*T);//1 — пустое дерево,0 — не пустое

void DelTree(Tree \*T);//удаление листа

#endif

Исходный файл Tree.cpp

#include "Tree.h"

#include "malloc.h"

short TreeError = TreeOk;

// инициализация — создается элемент, который бу-дет содержать корень дерева

void InitTree(Tree \*T) {

\*T = NULL;

TreeError = TreeOk;

}

//создание корня

void CreateRoot(Tree \*T) {

\*T = (Tree)malloc(sizeof(elementT));

if (\*T == NULL) {//если не удалось выделить память

TreeError = TreeNotMem;

return;

}

(\*T)->data = NULL;

(\*T)->LSon = NULL;

(\*T)->RSon = NULL;

}

//запись данных

void WriteDataTree(Tree \*T, basetype E) {

(\*T)->data = E;

}

//чтение

basetype ReadDataTree(Tree \*T) {

return (\*T)->data;

}

//1 — есть левый сын, 0 — нет

int IsLSon(Tree \*T) {

return ((\*T)->LSon != NULL) ? 1 : 0;

}

//1 — есть правый сын, 0 — нет

int IsRSon(Tree \*T) {

return ((\*T)->RSon != NULL) ? 1 : 0;

}

// перейти к левому сыну, где T — ад-рес ячейки, содержащей адрес текущей вершины, TS — адрес ячейки, со-держащей адрес корня левого поддерева(левого сына)

void MoveToLSon(Tree \*T, Tree \*TS) {

if ((\*T)->LSon == NULL) {

TreeError = NullPoint;

return;

}

\*TS = (\*T)->LSon;

}

//перейти к правому сыну

void MoveToRSon(Tree \*T, Tree \*TS) {

if ((\*T)->RSon == NULL) {

TreeError = NullPoint;

return;

}

\*TS = (\*T)->RSon;

}

//1 — пустое дерево,0 — не пустое

int IsEmptyTree(Tree \*T) {

return (\*T == NULL) ? 1 : 0;

}

//удаление листа

void DelTree(Tree \*T) {

free(\*T);

}

Код программы

#include "Tree.h"

#include "Stack.h"

#include <stdio.h>

#include <stdlib.h>

//Calc

Tree BildTree();////Строит дерево арифметического выражения, заданного в ОПЗ

void WritePrefix(Tree path);//вывод ППЗ

int Calc(Tree T);

int main() {

Tree T = BildTree();

WritePrefix(T);

printf("\n");

printf("%d", Calc(T));

getchar();

return 0;

}

//Строит дерево арифметического выражения, заданного в ОПЗ

Tree BildTree() {

Stack S;

basetype E;

Tree T;

char\* cha;

Tree path = NULL;

InitTree(&T);

InitStack(&S);

char c = getchar();

while (c != '\n') {

cha = (char\*)malloc(sizeof(char));

\*cha = c;

//path = (ptrel)malloc(sizeof(elementT));//выделить память корню

CreateRoot(&path);//создаем корень

WriteDataTree(&path, cha);//задаем значение корню

if ((c == '+') || (c == '-') || (c == '\*') || (c == 'd')) {

path->RSon = (ptrel)GetStack(&S);

path->LSon = (ptrel)GetStack(&S);

PutStack(&S, path);

}

else

PutStack(&S, path);

c = getchar();

}

return path;

}

//вывод ППЗ

void WritePrefix(Tree path) {

if (path != NULL) {

char c = \*(char\*)path->data;

printf("%c", c);

WritePrefix(path->LSon);

WritePrefix(path->RSon);

}

}

int Calc(Tree T) {

if (\*(char\*)(T->data) == '0')

return 0;

if (\*(char\*)(T->data) == '1')

return 1;

if (\*(char\*)(T->data) == '2')

return 2;

if (\*(char\*)(T->data) == '3')

return 3;

if (\*(char\*)(T->data) == '4')

return 4;

if (\*(char\*)(T->data) == '5')

return 5;

if (\*(char\*)(T->data) == '6')

return 6;

if (\*(char\*)(T->data) == '7')

return 7;

if (\*(char\*)(T->data) == '8')

return 8;

if (\*(char\*)(T->data) == '9')

return 9;

if (\*(char\*)(T->data) == '+')

return Calc(T->LSon) + Calc(T->RSon);

if (\*(char\*)(T->data) == '-')

return Calc(T->LSon) - Calc(T->RSon);

if (\*(char\*)(T->data) == '\*')

return Calc(T->LSon) \* Calc(T->RSon);

if (\*(char\*)(T->data) == 'd')

return Calc(T->LSon) / Calc(T->RSon);

}



