Факультет прикладной математики и кибернетики

Кафедра информатики

ОТЧЕТ

ПО УЧЕБНО-ВЫЧИСЛИТЕЛЬНОЙ ПРАКТИКЕ

РАБОТА С ДИНАМИЧЕСКИМИ СТРУКТУРАМИ ДАННЫХ В С/С++

Выполнил: студент 16 группы Баранов К.С.

Руководитель практики: ст. преп. Мальцева Л.Ф.

2016 г.**Задание 1**

1. Задание

Вариант 3. Вычислить с заданной степенью точности ε > 0 по формуле Симпсона значение и

1. Описание алгоритма

Увеличивая циклически количество разбиений на 2 добиваемся достижения заданной точности значения интеграла. Для этого создаем функции для вычисления значения подынтегральной функции, вычисления интеграла и циклического вычисления интеграла для достижения точности.

Подынтегральная функция вычисляется по формуле Симпсона:   
   
где

1. Исходный код программы

#include <stdio.h>

#include <math.h>

typedef float(\*myfunc)(float);

typedef float(\*integr)(float, float, int, myfunc);

// Первый интеграл

float func(float x) {

return 1 / (sqrt(1 + pow(x, 4)));

}

// Второй интеграл

float func2(float x) {

return x;

}

// Вычисление одного интеграла

float methodSimpson(float a, float b, int n, myfunc f) {

float h = (b - a) / n;

float integr = f(a) / 2 + f(b) / 2;

if (n % 2 == 1) n++;

for (int i = 1; i < n; i++) {

integr += (i % 2 + 1) \* f(a + i \* h);

}

return 2 \* h \* integr / 3;

}

// Вычисление интеграла с переменным кол-вом шагов

float integralWithEpilon(float a, float b, float e, myfunc f, integr in) {

int n = 10;

float i1 = in(a, b, n, f), i2 = 0;

do {

i2 = i1;

n \*= 2;

i1 = in(a, b, n, f);

} while (abs(i2 – i1) > e);

return i1;

}

// Проверка по формуле Ньютона-Лейбница

float prov(float a, float b) {

return (b\*b - a\*a) / 2;

}

int main() {

float a, b, c, d, eps;

printf("Vvedite a, b, c, d:\n");

scanf("%f%f%f%f", &a, &b, &c, &d);

printf("Vvedite epsilon: ");

scanf("%f", &eps);

printf("My integral = %f\n", integralWithEpilon(a, b, eps, func, methodSimpson));

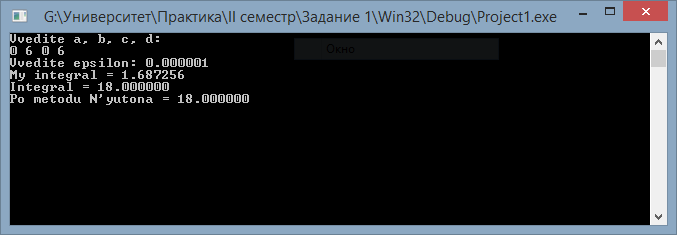
printf("Integral = %f\n", integralWithEpilon(c, d, eps, func2, methodSimpson));

printf("Po metodu N'yutona = %f", prov(c, d));

return 0;

}

1. Результаты расчета



**Задание 2**

1. Задание

Вариант 20. Реализовать операцию % (остаток от деления нацело) для двух длинных натуральных чисел. Операнды и результат должны быть представлены линейными двусвязными списками, информационная часть элементов которых две цифры числа.

1. Описание алгоритма

Вначале описана структура bigInt, которая будет создавать двусвязный список для хранения большого числа. Каждый элемент списка будет хранить две последовательные цифры числа.

Функция чтения большого числа из строки считывает посимвольно число и записывает его в список. Функция печати делает обратное преобразование.

Реализованы функции подсчета длины строки и длины числа. Последняя вычисляет количество парочек цифр в числе.

Функция сравнения больших чисел работает аналогично сравнению для строк, то есть посимвольно сравнивает до первого различия и возвращаем результат этого различия (какое число больше).

Удаление числа подразумевает под собой очистку памяти от всех элементов списка, хранящего число и присвоение концам списка значения NULL.

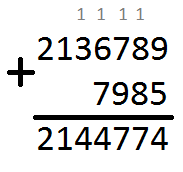
Функция добавления к началу списка создает новый элемент, помещает в него две цифры числа, связывает указателем с текущим головным элементом и заменяет собой головной элемент, становясь первым.

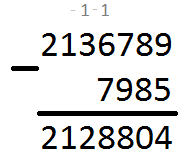
Добавление к концу списка предусматривает «сдвиг» числа внутри списка, если оно имеет ведущие нули в головном элементе. Если требуется добавить одну цифру в конец, сначала проверяется наличие ведущего нуля, если он есть, то всё число сдвигается на один символ вперед и цифра добавляется в конец, иначе создается новый головной элемент и в него сдвигаются элементы, чтобы «освободить» место для нового.

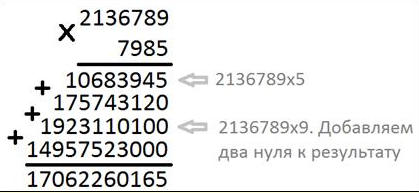
Функция получения цифры числа возвращает i-тую цифру числа путем последовательного поиска индекса (индекс первой цифры слева - 1).

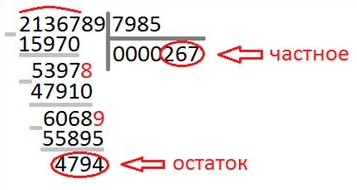
Функция удаления ведущих нулей призвана обеспечивать правильный вход данных и удаления лишних ведущих нулей, которые мешают корректному выполнению операций над числами.

Замена цифры в числе на другую цифру требуется для выполнения операций, когда на место нулей ставится цифра (-1), которая впоследствие заменяется на 0.

Принцип функции сложения больших чисел основан на сложении «столбиком». Складываем поразрядно числа в пределах длины меньшего числа, добавляя единицу, если сумма на предыдущем шаге была больше 9, находим остаток от деления на 10 и записываем его в разряд, над которым выполнялась операция. После того, как разряды у одного из чисел закончатся, переносим в сумму оставшиеся цифры из больших разрядов.

Реализация вычитания аналогична сложению: находим разности цифр одного разряда и не забываем про заимствование десятка из следующего разряда. После того, как закончатся разряды второго числа (оно меньше), переносим остальные цифры в результат.

После того, как были реализованы операции сложения и вычитания, перейдем к написанию операции умножения. В его основе лежит принцип умножения «столбиком». Выбираем одно из больших чисел, и поэтапно умножаем числа из каждого разряда этого большого числа, на другое большое число. В результате на каждом шаге у нас получаются «промежуточные» большие числа, суммируя которые с помощью описанной выше функции суммы получаем необходимый результат. При этом не забываем в зависимости от разряда множителя «добивать» начальные разряды «промежуточных» больших чисел нулями. Как и в предыдущих операциях при поразрядном перемножении в результат мы записываем остаток от деления на 10 и, если результат поразрядного перемножения больше либо равен 10, то на следующем шаге перемножения мы прибавляем число равное целой части от деления предыдущего результата на 10.

Пожалуй, самой сложной для реализации, является операция деления и нахождение остатка от деления двух больших чисел. В основе лежит принцип деления «столбиком». Ход алгоритма следующий: сравниваем делитель с делимым, прибавляя поразрядно по одной цифре к делителю в случае, если получившийся делитель меньше делимого, при этом в частное записываем 0. На рисунке 8 видно этот этап: 2<7985, в частное записываем 0, затем 21<7985, в частное записываем 0, и так далее пока не поменяется знак неравенства 21367>7985. После этого запускается цикл по нахождению следующей цифры частного. На каждом шаге делитель прибавляется на величину равную самому делителю, пока он не станет больше либо равен нашему промежуточному делимому, т.е. 21367. Шаг цикла, на котором выполнится данное условие, и будет искомой цифрой для частного. Затем вычитаем из промежуточного делимого полученное в ходе цикла число и получаем промежуточный остаток. Так как он точно меньше делителя (в связи с предыдущими условиями), добавляем к нему следующую не задействованную цифру делимого и переходим к первому шагу алгоритма. Алгоритм считается выполненным, если получается остаток, меньший делителя и не осталось ни одной незадействованной цифры делимого.

1. Исходный код программы

#include <stdio.h>

struct bigInt { // структура большого числа

short d1;

short d2;

bigInt \*prev;

bigInt \*next;

};

int my\_strlen (const char \*str) { // Подсчет длины строки

int i = 0;

while (\*str++) i++;

return i;

}

int bigintlength(bigInt \*&hbi) { // Подсчет длины числа

bigInt \*cur = hbi;

int i = 0;

while (cur) {

i++;

cur = cur->next;

}

return i;

}

// Сравнение больших чисел

int bigintcmp(bigInt \*&hbi1, bigInt \*&tbi1, bigInt \*&hbi2, bigInt \*&tbi2) {

int ch1 = bigintlength(hbi1);

int ch2 = bigintlength(hbi2);

bigInt \*cur1 = hbi1;

bigInt \*cur2 = hbi2;

while (ch1 == ch2 && cur1 != NULL && cur2 != NULL) {

ch1 = cur1->d2 + cur1->d1 \* 10;

ch2 = cur2->d2 + cur2->d1 \* 10;

cur1 = cur1->next;

cur2 = cur2->next;

}

return ch1 - ch2;

}

// Инициализация

void init(bigInt \*&hbi, bigInt \*&tbi) {

hbi = NULL;

tbi = NULL;

}

// Удаление большого числа

void deleteBigInt(bigInt \*&hbi, bigInt \*&tbi) {

bigInt \*prev = NULL;

while (hbi != NULL) {

prev = hbi;

hbi = hbi->next;

if (hbi) hbi->prev = NULL;

delete prev;

}

tbi = NULL;

}

// Замена в числе цифр «a» на «b»

void replaceDigit(short a, short b, bigInt \*&hbi, bigInt \*&tbi) {

bigInt \*cur = hbi;

while (cur!= NULL) {

if (cur->d1 == a) cur->d1 = b;

if (cur->d2 == a) cur->d2 = b;

cur = cur->next;

}

}

// Удаление ведущих нулей

void deleteZero(bigInt \*&hbi, bigInt \*&tbi) {

bigInt \*temp = NULL;

while (hbi != NULL && hbi->next != NULL) {

if (hbi->d1 == 0 && hbi->d2 == 0) {

temp = hbi;

hbi = hbi->next;

hbi->prev = NULL;

delete temp;

} else break;

}

}

// Добавление к началу числа

void addToHead(bigInt \*&hbi, bigInt \*&tbi, int inf1, int inf2) {

bigInt \* newBi = new bigInt;

int z = 0;

newBi->d1 = inf1;

newBi->d2 = inf2;

newBi->next = hbi;

newBi->prev = NULL;

if (inf1 == 0) z++;

if (inf2 == 0) z++;

if (hbi == NULL) {

hbi = newBi;

tbi = newBi;

} else {

hbi->prev = newBi;

hbi = newBi;

}

}

// Добавление к концу числа

void addToTale(bigInt \*&hbi, bigInt \*&tbi, int inf1, int inf2) {

if (tbi != NULL) {

if (inf1 == 0 && inf2 != 0 && hbi->d1 != 0) {

bigInt \*newBi = new bigInt;

newBi->d1 = 0; newBi->d2 = 0;

newBi->next = hbi;

newBi->prev = NULL;

hbi->prev = newBi;

hbi = newBi;

}

if (inf1 == 0 && inf2 != 0 && hbi->d1 == 0) {

bigInt \*cur = hbi;

bigInt \*prev = NULL;

while (cur) {

cur->d1 = cur->d2;

prev = cur;

cur = cur->next;

if (cur != NULL) prev->d2 = cur->d1;

}

prev->d2 = inf2;

return;

}

}

bigInt \*newBi = new bigInt;

newBi->d1 = inf1; newBi->d2 = inf2;

newBi->next = NULL;

newBi->prev = tbi;

if (tbi == NULL) {

hbi = newBi;

tbi = newBi;

} else {

tbi->next = newBi;

tbi = newBi;

}

}

// Печать большого числа

void printBigInt(bigInt \*&head) {

bigInt \*cur = head;

bool flag = false;

if (cur == NULL) {

printf("0");

} else

while (cur != NULL) {

printf("%d%d", cur->d1, cur->d2);

cur = cur->next;

}

printf("\n");

}

// Получить n-ую цифру числа

int getDigit(bigInt \*&head, int n) {

if (n > bigintlength(head)\*2) {

return -1;

}

int k = (n-1) / 2;

int i = 0;

bigInt \*cur = head;

for (; i < k; i++) {

cur = cur->next;

}

return (n % 2 > 0) ? cur->d1 : cur->d2;

}

// Преобразование из строки в список

void strToBigint(char \*chislo, bigInt \*&hbi, bigInt \*&tbi) {

int n = my\_strlen(chislo);

int ch1 = 0, ch2 = 0;

for (int i = n-1; i >= 0; i-=2) {

ch2 = (chislo[i] - '0');

if (chislo[i-1] - '0' > 0)

ch1 = (chislo[i-1] - '0');

addToHead(hbi, tbi, ch1, ch2);

ch1 = ch2 = 0;

}

deleteZero(hbi, tbi);

}

// Разность двух больших чисел

int raznost(bigInt \*&hbi1, bigInt \*&tbi1, bigInt \*&hbi2, bigInt \*&tbi2, bigInt \*&hbi3, bigInt \*&tbi3) {

if (bigintcmp(hbi1, tbi1, hbi2, tbi2) < 0) {

addToHead(hbi3, tbi3, 0, 0);

return 0;

}

bigInt \*cur1 = tbi1;

bigInt \*cur2 = tbi2;

int n1 = 0, n2 = 0, perenos = 0;// ch1 = 0, ch2 = 0;

while (cur1 != NULL && cur2 != NULL) {

n1 = cur1->d2 - cur2->d2 - perenos;

perenos = 0;

if (n1 < 0) {

n1 = 10 + n1; perenos = 1;

}

n2 = cur1->d1 - cur2->d1 - perenos;

perenos = 0;

if (n2 < 0) {

n2 = 10 + n2; perenos = 1;

}

addToHead(hbi3, tbi3, n2, n1);

n1 = n2 = 0;

cur1 = cur1->prev;

cur2 = cur2->prev;

}

while (cur1 != NULL) {

n1 = cur1->d2 - perenos;

perenos = 0;

if (n1 < 0) {

n1 = 10 + n1;

perenos = 1;

}

n2 = cur1->d1 - perenos;

perenos = 0;

if (n2 < 0) {

n2 = 10 + n2;

perenos = 1;

}

if (((n1 || n2) && cur1->prev == NULL) || cur1->prev != NULL) addToHead(hbi3, tbi3, n2, n1);

cur1 = cur1->prev;

}

deleteZero(hbi3, tbi3);

return 0;

}

// Сумма больших чисел

void summa(bigInt \*&hbi1, bigInt \*&tbi1, bigInt \*&hbi2, bigInt \*&tbi2, bigInt \*&hbi3, bigInt \*&tbi3) {

// Поменять местами, чтбы большее число было верхним

bigInt \*cur1 = tbi1;

bigInt \*cur2 = tbi2;

int n1 = 0, n2 = 0, perenos = 0, ch = 0;

while (cur1 != NULL && cur2 != NULL) {

n1 = cur1->d2 + cur2->d2 + perenos;

perenos = 0;

if (n1 > 9) {

n1 = n1 - 10;

perenos = 1;

}

n2 = cur1->d1 + cur2->d1 + perenos;

perenos = 0;

if (n2 > 9) {

n2 = n2 - 10;

perenos = 1;

}

addToHead(hbi3, tbi3, n2, n1);

n1 = n2 = ch = 0;

cur1 = cur1->prev;

cur2 = cur2->prev;

}

while (cur1 != NULL) {

n1 = cur1->d2 + perenos;

n2 = cur1->d1;

perenos = 0;

addToHead(hbi3, tbi3, n2, n1);

cur1 = cur1->prev;

}

while (cur2 != NULL) {

n1 = cur2->d2 + perenos;

n2 = cur2->d1;

perenos = 0;

addToHead(hbi3, tbi3, n2, n1);

cur2 = cur2->prev;

}

if (perenos) {

addToHead(hbi3, tbi3, 0, perenos);

}

}

// Произведение больших чисел

void proizved(bigInt \*&hbi1, bigInt \*&tbi1, bigInt \*&hbi2, bigInt \*&tbi2, bigInt \*&hbi3, bigInt \*&tbi3) {

if (hbi1->d2 == 0 || hbi2->d2 == 0) { // Проверка на ноль

addToHead(hbi3, tbi3, 0, 0);

}

int poryadok = 0, perenos = 0; // Кол-во нулей для сложения

int n1 = 0, n2 = 0, n = 0, ch = 0, i; bigInt \*cur1 = tbi1;

bigInt \*cur2 = tbi2;

bigInt \*htemp; bigInt \*ttemp;

init(htemp, ttemp);

bigInt \*htemp2; bigInt \*ttemp2;

init(htemp2, ttemp2);

while (cur2 != NULL) {

for (i = poryadok; i > 1; i -= 2) {

addToHead(htemp2, ttemp2, 0, 0);

}

n = cur2->d2;

while (cur1 != NULL) {

n1 = cur1->d2 \* n + perenos;

perenos = 0;

if (n1 > 9) {

perenos = n1 / 10; n1 %= 10;

}

n2 = cur1->d1 \* n + perenos;

perenos = 0;

if (n2 > 9) {

perenos = n2 / 10; n2 %= 10;

}

addToHead(htemp2, ttemp2, n2, n1);

cur1 = cur1->prev;

}

if (perenos) {

addToHead(htemp2, ttemp2, 0, perenos);

perenos = 0;

}

summa(htemp, ttemp, htemp2, ttemp2, hbi3, tbi3);

deleteBigInt(htemp, ttemp);

deleteBigInt(htemp2, ttemp2);

htemp = hbi3; ttemp = tbi3;

hbi3 = NULL; tbi3 = NULL;

cur1 = tbi1;

poryadok++;

for (i = poryadok; i > 1; i -= 2) {

addToHead(htemp2, ttemp2, 0, 0);

}

n = cur2->d1;

n1 = 0;

while (cur1 != NULL) {

n2 = cur1->d2 \* n + perenos;

perenos = 0;

if (n2 > 9) {

perenos = n2 / 10; n2 %= 10;

}

addToHead(htemp2, ttemp2, n2, n1);

n1 = cur1->d1 \* n + perenos;

perenos = 0;

if (n1 > 9) {

perenos = n1 / 10; n1 %= 10;

}

cur1 = cur1->prev;

}

if (n1 || perenos) {

addToHead(htemp2, ttemp2, perenos, n1);

perenos = 0;

}

cur1 = tbi1; poryadok++;

summa(htemp, ttemp, htemp2, ttemp2, hbi3, tbi3);

deleteBigInt(htemp, ttemp);

deleteBigInt(htemp2, ttemp2);

htemp = hbi3; ttemp = tbi3;

hbi3 = NULL; tbi3 = NULL;

cur2 = cur2->prev;

}

hbi3 = htemp; tbi3 = ttemp;

}

// Частное больших чисел

void delenie(bigInt \*&hbi1, bigInt \*&tbi1, bigInt \*&hbi2, bigInt \*&tbi2, bigInt \*&hchast, bigInt \*&tchast, bigInt \*&host, bigInt \*&tost) {

int n = bigintlength(hbi1) \* 2;

int razr = 1, delenie = 0, k = 0, sb = 0;

while (1) {

while (bigintcmp(host, tost, hbi2, tbi2) < 0) {

k = getDigit(hbi1, razr);

if (k >= 0) {

addToTale(host, tost, 0, k);

razr++; sb++;

if (sb > 1) {

addToTale(hchast, tchast, 0, -1);

}

} else {

replaceDigit(-1, 0, hchast, tchast);

deleteZero(hchast, tchast);

deleteZero(host, tost);

return;

}

}

while (bigintcmp(host, tost, hbi2, tbi2) >= 0) {

bigInt \*htemp, \*ttemp;

init(htemp, ttemp);

raznost(host, tost, hbi2, tbi2, htemp, ttemp);

//printBigInt(htemp);

deleteBigInt(host, tost);

host = htemp; tost = ttemp;

htemp = NULL; ttemp = NULL;

delenie++;

}

addToTale(hchast, tchast, 0, delenie);

delenie = 0; sb = 0;

}

}

// Главная функция

int main() {

char \*str1 = new char[255]; // Ввод

char \*str2 = new char[255];

printf("1 chislo:\n");

gets(str1);

printf("2 chislo:\n");

gets(str2);

bigInt \*hbi1, \*tbi1; // Инициализация

init(hbi1, tbi1);

bigInt \*hbi2, \*tbi2;

init(hbi2, tbi2);

bigInt \*hbi3, \*tbi3;

init(hbi3, tbi3);

bigInt \*hbi4, \*tbi4;

init(hbi4, tbi4);

strToBigint(str1, hbi1, tbi1); // Запись

strToBigint(str2, hbi2, tbi2);

printf("\n");

// Выполнение операций

summa(hbi1, tbi1, hbi2, tbi2, hbi3, tbi3);

printf("Summa: ");

printBigInt(hbi3);

deleteBigInt(hbi3, tbi3);

raznost(hbi1, tbi1, hbi2, tbi2, hbi3, tbi3);

printf("Raznost: ");

printBigInt(hbi3);

deleteBigInt(hbi3, tbi3);

proizved(hbi1, tbi1, hbi2, tbi2, hbi3, tbi3);

printf("Proizvedenie: ");

printBigInt(hbi3);

deleteBigInt(hbi3, tbi3);

if (hbi2->d2 == 0) {

printf("Delenie nevozmojno!\n");

} else {

delenie(hbi1, tbi1, hbi2, tbi2, hbi3, tbi3, hbi4, tbi4);

printf("Chastnoe: ");

printBigInt(hbi3);

printf("Ostatok: ");

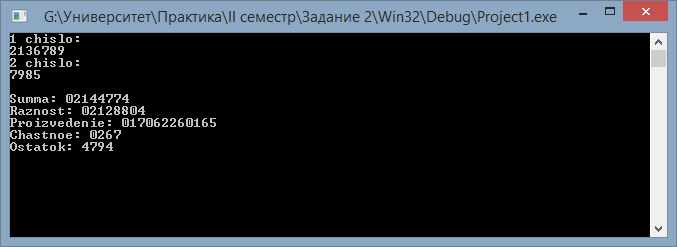
printBigInt(hbi4);

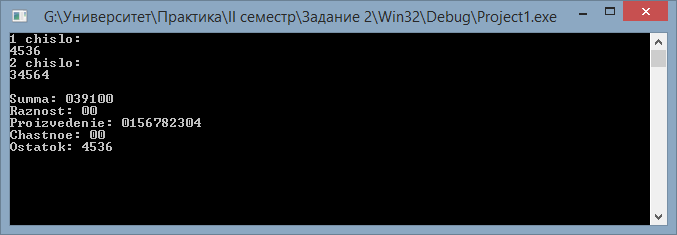
}

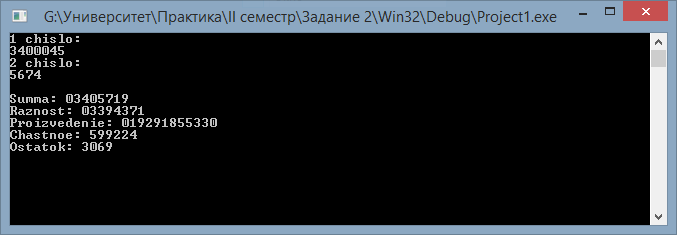
return 0;

}

1. Результат расчета







**Задание 3**

1. Задание

12А. На входе дана произвольная последовательность чисел . Написать программу, которая:

* строит бинарное дерево поиска;
* печатает минимальные элементы каждого уровня снизу-вверх (обход снизу-вверх по уровням).

Последовательность чисел задана в текстовом файле.

1. Описание алгоритма

В программе описана структура для дерева и функции для работы с ним:

* Инициализация дерева;
* Проверка дерева на пустоту;
* Добавление узла в дерево;
* Поиск заданного узла в дереве;
* Удаление из дерева заданного узла;
* Удаление из дерева заданного поддерева;
* Обход дерева по заданному выше правилу.

Для обхода дерева понадобилась такая динамическая структура как стек. Для его реализации созданы функции добавления элемента, извлечения элемента, печать стека без верхнего элемента.

В основе алгоритма заданного обхода лежит алгоритм обхода дерева в ширину. Изначально созданы две очереди: для обхода и для сохранения вершин нижнего уровня. В цикле происходит обход узлов первой очереди с сохранением их потомков во вторую очередь, среди которых, затем, ищется минимальное значение, потомки копируются в первую очередь и процесс повторяется. Все минимальные значения сохраняются в стек, чтобы в конце получился ответ.

1. Исходный код программы

#include <stdio.h>

struct tree {

int inf;

bool fl;

tree \*ls, \*rs;

};

struct List {

tree \*unit;

List \*next;

};

struct Stack {

int info;

Stack \*next;

};

template <class T> // Инициализация

void init(T \*&head) {

head = NULL;

}

template <class T> // Проверка на пустоту

bool isEmpty(T \*&head) {

return (head == NULL) ? true : false;

}

void addTree(tree \*&root, int info) { // Добавление узла

tree \*nt = new tree;

nt->inf = info;

nt->fl = false;

nt->ls = nt->rs = NULL;

if (isEmpty(root)) {

root = nt;

} else {

tree \*cur = root;

tree \*prev = NULL;

while (cur != NULL) {

prev = cur;

if (info > cur->inf) cur = cur->rs;

else cur = cur->ls;

}

if (info > prev->inf) prev->rs = nt;

else prev->ls = nt;

}

}

// Поиск узла в дереве

void findTree(tree \*&root, int info, tree \*&ad, tree \*&adr) {

ad = root;

adr = NULL;

while (ad != NULL && ad->inf != info) {

adr = ad;

if (info > ad->inf) ad = ad->rs;

else ad = ad->ls;

}

if (ad == NULL) adr = NULL;

}

// Удаление узла

void delTreeUz(tree \*&root, int info) {

tree \*cur, \*prev;

findTree(root, info, cur, prev);

if (cur != NULL) {

// Если корень

if (cur == root) {

if (cur->ls == NULL && cur->rs == NULL) {

root = NULL;

}

if (cur->ls != NULL && cur->rs == NULL) {

root = cur->ls;

delete root;

}

if (cur->rs != NULL && cur->ls == NULL) {

root = cur->rs;

delete cur;

}

return;

}

// Если внутр. с двумя потомками

if (cur->ls != NULL && cur->rs != NULL) {

tree \*temp = cur;

tree \*temp2 = cur->rs;

while (temp2->ls != NULL) {

temp = temp2;

temp2 = temp2->ls;

}

cur->inf = temp2->inf;

if (isEmpty(cur->rs->ls)) {

cur->rs = temp2->rs;

delete temp2;

} else

delTreeUz(cur->rs, cur->inf);

return;

}

// Если лист

if (cur->ls == NULL && cur->rs == NULL) {

if (cur == prev->rs) prev->rs = NULL;

else prev->ls = NULL;

delete cur;

return;

}

// Если внутр. узел с одним потомком

if (cur->ls == NULL && cur->rs != NULL) {

prev->rs = cur->rs;

delete cur;

return;

}

if (cur->ls != NULL && cur->rs == NULL) {

prev->ls = cur->ls;

delete cur;

return;

}

}

}

// Рекурсия удаления подерева

void freemem(tree \*&root, char priz) {

if (root != NULL) {

if (priz == 'R') {

freemem(root->rs, priz);

} else if (priz == 'L') {

freemem(root->ls, priz);

}

delete root;

}

}

// Удаление поддерева

void delTreeTree(tree \*&root, int info, char priz) {

tree \*cur = NULL, \*prev = NULL;

findTree(root, info, cur, prev);

freemem(cur, priz);

}

// Чтение элементов дерева из файла

tree \*readTreeFromFile(char \*fname) {

FILE \*in = fopen(fname, "r");

tree \*root;

init(root);

int a = 0;

while (fscanf(in, "%d", &a) > 0) {

addTree(root, a);

}

fclose(in);

return root;

}

// Добавление в конец списка

void addToListTail(List \*&head, tree \*&unit) {

List \*el = new List;

el->unit = unit;

el->next = NULL;

List \*cur = head;

List \*prev = NULL;

while (!isEmpty(cur)) {

prev = cur;

cur = cur->next;

}

if (isEmpty(head)) {

head = el;

} else {

prev->next = el;

}

}

// Удаление элемента из начала списка

tree \*delFromListHead(List \*&head) {

List \*cur = head;

head = head->next;

tree \*el = cur->unit;

delete cur;

return el;

}

// Удаление элемента из конца списка

tree \*delFromListTail(List \*&head) {

if (isEmpty(head)) {

return NULL;

}

List \*cur = NULL;

List \*prev = NULL;

List \*next = head;

tree \*el = NULL;

while (!isEmpty(next)) {

prev = cur;

cur = next;

next = next->next;

}

if (isEmpty(prev)) {

el = cur->unit;

head = NULL;

} else {

el = cur->unit;

prev->next = next;

}

delete cur;

return el;

}

// Копирование списка

void copyList(List \*&head1, List \*&head2) {

List \*cur = head1;

while (!isEmpty(cur)) {

addToListTail(head2, cur->unit);

cur = cur->next;

}

}

// Поиск минимума в списке

int getMin(List \*&head) {

List \*cur = head;

if (isEmpty(head)) return 0;

int min = cur->unit->inf;

while (cur != NULL) {

if (cur->unit->inf < min) {

min = cur->unit->inf;

}

cur = cur->next;

}

return min;

}

// поместить элемент в стек

void push(Stack \*&st, int n) {

Stack \*el = new Stack;

el->info = n;

el->next = st;

st = el;

}

// Извлечь элемент из стека

int pop(Stack \*&st) {

Stack \*cur = st;

int k = cur->info;

st = st->next;

delete cur;

return k;

}

// Печать стека без первого элемента

void printStackWithoutFirst(Stack \*&st) {

Stack \*cur = st;

int a = pop(st);

while (!isEmpty(cur->next)) {

a = pop(st);

printf("%d ", a);

cur = cur->next;

}

}

// Инфиксный обход дерева

void infixObhod(tree \*&root) {

List \*st = NULL;

tree \*branch = NULL;

addToListTail(st, root);

while (!isEmpty(st)) {

branch = delFromListTail(st);

if (!isEmpty(branch)) {

if (isEmpty(branch->ls) && isEmpty(branch->rs)) {

printf(" %d ", branch->inf);

branch->fl = false;

} else {

if (branch->fl) {

printf(" %d ", branch->inf);

branch->fl = false;

} else {

branch->fl = true;

addToListTail(st, branch->rs);

addToListTail(st, branch);

addToListTail(st, branch->ls);

}

}

}

}

}

// Обход в ширину с выводом минимальных элементов уровней

void width(tree \*&root) {

if (isEmpty(root))

return;

List \*hqueue1 = NULL;

List \*hqueue2 = NULL;

Stack \*st = NULL;

addToListTail(hqueue1, root);

addToListTail(hqueue2, root);

push(st, root->inf);

do {

while (!isEmpty(hqueue1)) {

tree \*cur = delFromListHead(hqueue1);

delFromListHead(hqueue2);

if (cur->ls) {

addToListTail(hqueue2, cur->ls);

}

if (cur->rs) {

addToListTail(hqueue2, cur->rs);

}

}

int k = getMin(hqueue2);

push(st, k);

copyList(hqueue2, hqueue1);

} while (!isEmpty(hqueue2));

printStackWithoutFirst(st);

}

int main() {

char fname[100];

printf("Enter the file name: \n");

gets(fname);

printf("Begin to bild a tree...\n");

tree \*my\_tree = readTreeFromFile(fname);

printf("Infix traversal of the tree:\n");

infixObhod(my\_tree);

printf("\n");

printf("Result bypass: \n");

width(my\_tree);

printf("\n");

char c;

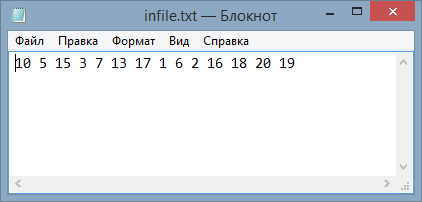
scanf("%c", &c);

return 0;

}

1. Результат расчета

Содержимое текстового файла infile.txt:



Дерево:

Результат работы программы:

