1. **Строки**

В билете будет 3 задачи на строки. Примеры задач в файле «Задачи Обработка строк».

#include <iostream>

#include <locale>

#include <string>

#include <windows.h>

using namespace std;

int main()

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

setlocale(LC\_ALL, "Russian");

string s;

getline(cin, s);

for (int i = 0; i < s.size(); i++)

**size**

Кроме того, у строк есть метод length(), который также возвращает длину строки.

**resize**

S.resize(n) - Изменяет длину строки, новая длина строки становится равна n. При этом строка может как уменьшится, так и увеличиться. Если вызвать в виде S.resize(n, c), где c - символ, то при увеличении длины строки добавляемые символы будут равны c.

**clear**

S.clear() - очищает строчку, строка становится пустой.

**empty**

S.empty() - возвращает true, если строка пуста, false - если непуста.

**push\_back**

S.push\_back(c) - добавляет в конец строки символ c, вызывается с одним параметром типа char.

**append**

Добавляет в конец строки несколько символов, другую строку или фрагмент другой строки. Имеет много способов вызова.

S.append(n, c) - добавляет в конец строки n одинаковых символов, равных с. n имеет целочисленный тип, c - char.

S.append(T) - добавляет в конец строки S содержимое строки T. T может быть объектом класса string или C-строкой.

S.append(T, pos, count) - добавляет в конец строки S символы строки T начиная с символа с индексом pos количеством count.

**erase**

S.erase(pos) - удаляет из строки S  с символа с индексом pos и до конца строки.

S.erase(pos, count) - удаляет из строки S  с символа с индексом pos количеством count или до конца строки, если pos + count > S.size().

**insert**

Вставляет в середину строки несколько символов, другую строку или фрагмент другой строки

S.insert(i, n, c) - вставить n одинаковых символов, равных с. n имеет целочисленный тип, c - char.

S.insert(i, T) - вставить содержимое строки T. T может быть объектом класса string или C-строкой.

S.insert(i, T, pos, count) - вставить символы строки T начиная с символа с индексом pos количеством count.

**substr**

S.substr(pos) - возвращает подстроку данной строки начиная с символа с индексом pos и до конца строки.

S.substr(pos, count) - возвращает подстроку данной строки начиная с символа с индексом pos количеством count или до конца строки, если pos + count > S.size().

**replace**

Заменяет фрагмент строки на несколько равных символов, другую строку или фрагмент другой строки.

S.replace(pos, count, n, c) - вставить n одинаковых символов, равных с. n имеет целочисленный тип, c - char.

S.replace(pos, count, T) - вставить содержимое строки T. T может быть объектом класса string или C-строкой.

S.replace(pos, count, T, pos2, count2) - вставить символы строки T начиная с символа с индексом pos количеством count.

**find**

Ищет в данной строке первое вхождение другой строки str. Возвращается номер первого символа, начиная с которого далее идет подстрока, равная строке str. Если эта строка не найдена, то возвращается константа string::npos (которая равна -1, но при этом является беззнаковой, то есть на самом деле является большим безннаковым положительным числом).

Если задано значение pos, то поиск начинается с позиции pos, то есть возращаемое значение будет не меньше, чем pos. Если значение pos не указано, то считается, что оно равно 0 - поиск осуществляется с начала строки.

S.find(str, pos = 0) - искать первое входение строки str начиная с позиции pos. Если pos не задано - то начиная с начала строки S.

S.find(str, pos, n) - искать в данной строке подстроку, равную первым n символам строки str. Значение pos должно быть задано.

**rfind**

Ищет последнее вхождение подстроки ("правый" поиск). Способы вызова аналогичны способам вызова метода find.

**find\_first\_of**

Ищет в данной строке первое появление любого из символов данной строки str. Возвращается номер этого символа или значение string::npos.

Если задано значение pos, то поиск начинается с позиции pos, то есть возращаемое значение будет не меньше, чем pos. Если значение pos не указано, то считается, что оно равно 0 - поиск осуществляется с начала строки.

S.find\_first\_of(str, pos = 0) - искать первое входение любого символа строки str начиная с позиции pos. Если pos не задано - то начиная с начала строки S.

**find\_last\_of**

Ищет в данной строке последнее появление любого из символов данной строки str. Способы вызова и возвращаемое значение аналогичны методу find\_first\_of.

**find\_first\_not\_of**

Ищет в данной строке первое появление символа, отличного от символов строки str. Способы вызова и возвращаемое значение аналогичны методу find\_first\_of.

**find\_last\_not\_of**

Ищет в данной строке последнее появление символа, отличного от символов строки str. Способы вызова и возвращаемое значение аналогичны методу find\_first\_of.

1. **Файлы**

Задача на работу с файлами.

- Похожие есть в задачнике «Записи и файлы» (задачи с 17 по 21)

- Могут быть задачи на А/Е – свойства (Пример из лекции 27)

struct student{};

struct exam{ student st[20];};

int main(){

exam s;

ifstream in("17.txt");

ofstream out("17.bin", ios::binary);

while (in >> s.num >> s.subject >> s.teacher >> s.data) {

for (int i = 0; i < 20; i++){

in >> s.st[i].surname >> s.st[i].mark; }

out.write((char\*)&s, sizeof(exam));}

in.close();

out.close();

ifstream in1("17.bin", ios::binary);

while (in1.read((char\*)&s, sizeof(exam))){

for (int i = 0; i < 20; i++)

if (s.st[i].mark <= 55) {

if (s.subject[0] == 'И') }

void k(ifstream& in1, ifstream& in2, ofstream& out1){

Postavki a;

SvedeniyaODetalyah b;

in2.read((char\*)&b, sizeof(SvedeniyaODetalyah));

while (in1.read((char\*)&a, sizeof(Postavki))){

if (a.n1 == b.n1s) {

ResultRecord c;

strcpy\_s(c.namer, b.name);

out1.write((char\*)&c, sizeof(ResultRecord));}

else{

in2.read((char\*)&b, sizeof(SvedeniyaODetalyah)); {

ResultRecord c;

out1.write((char\*)&c, sizeof(ResultRecord));}}}

void a(ifstream& in1, ifstream& in2, ofstream& out)

{

person p;

while (in2.peek() != EOF)

{

in2.read((char\*)&p, sizeof(p));

in1.clear();

in1.seekg(0, ios::beg);

sub s;

while (in1.peek() != EOF)

{

in1.read((char\*)&s, sizeof(s));

if (strcmp(p.surname, s.surname) == 0)

{

result\_a r;

strcpy\_s(r.street, p.street);

r.h = p.h;

r.f = p.f;

strcpy\_s(r.izd\_name, s.name\_izd);

out.write((char\*)&r, sizeof(r));

}

}

}

in1.close();

in2.close();

out.close();

}

{

return strcmp(a.town, b.town) == 0;

}

bool kamp(typeStruct l, typeStruct f, typeStruct m)

{

return l.age > f.age && l.age > m.age;

}

void formNewBin(ifstream & fM, ifstream & fF, ifstream & fL, ofstream & gM)

{

typeStruct xM, xF, xL;

bool b, d;

fM.read((char\*)&xM, sizeof(typeStruct));

while (!fM.eof())

{

b = false;

while (fF.peek() != EOF && !b)

{

fF >> xF.fam >> xF.age >> xF.town;

if (sameTown(xM, xF))

{

d = true;

while (fL.peek() != EOF && d)

{

fL >> xL.fam >> xL.age >> xL.town;

if (sameTown(xM, xL))

if (!kamp(xL, xF, xM))

d = false;

}

b = d;

fL.clear();

fL.seekg(0);

}

}

fF.clear();

fF.seekg(0);

if (b)

gM << xM.fam << ' ' << xM.age << ' ' << xM.town << endl;

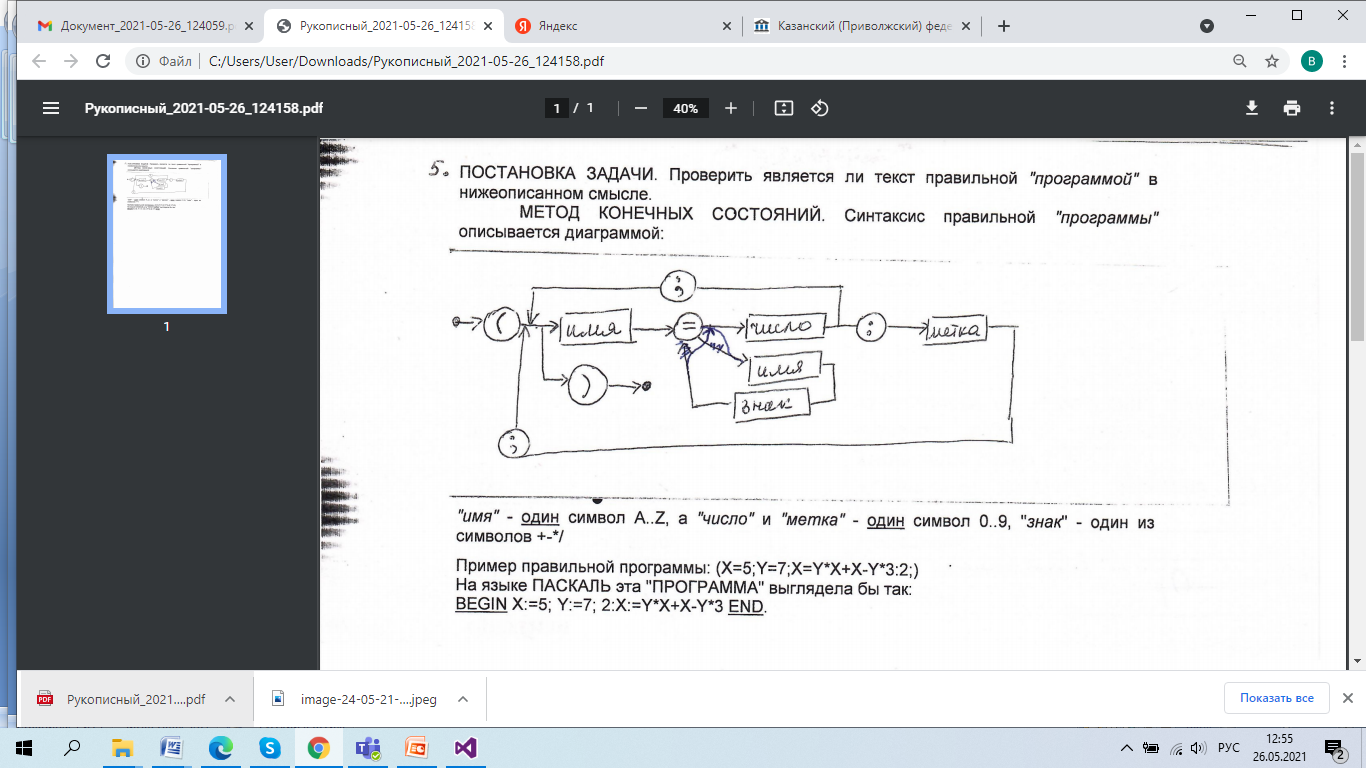
fM.read((char\*)&xM, sizeof(typeStruct));

}

}

1. **Метод конечных состояний.**

В билете будет одна задача. Примерно такого характера.



Нужно будет построить автомат, запрограммировать его и протестировать код.

{ char x;

int q = 0;

ifstream f ("Text.txt");

while (f.peek()!=EOF )

{

f >> x;

switch (q)

{

case 0: if (x == '{')

q = 1;

else

if (x == '"')

{

cout << x;

q = 2;

}

else cout << x;

break;

case 1: if (x == '}')

q = 0;

break;

case 2: if (x == '"')

{

cout << x;

q = 0;

}

else cout << x;

break;

}

}

void main()

{

char x;

int q = 1, k = 0, l = 0;

ifstream f("Text.txt");

//char s[20] = { " fjhjlyf 1276 hрk " };

while (f.peek() != EOF)

{

f.get(x);

switch (q) {

case 1: {

if (x == ' ') q = 1;

else if (x >= '0'&& x <= '9') q = 2;

else if ((x >= 'a'&& x <= 'z') || (x >= 'A'&& x <= 'Z'))

q = 3;

else q=4;

break;

}

case 2: {

if (x >= '0'&& x <= '9') q = 2;

else if (x == ' ')

{

k++;

q = 1;

}

else q = 4;

break;

}

case 3: {

if ((x >= 'a'&& x <= 'z') || (x >= 'A'&& x <= 'Z'))

q = 3;

else if (x == ' ')

{

l++;

q = 1;

}

else q = 4;

break;

}

case 4: if (x == ' ') q = 1;

else q = 4;

}

}

cout << "number of words: "<<l << endl;

cout << "number of digits: " << k << endl;

system("pause");

}

int main()

{ setlocale (LC\_ALL, "Russian");

char s[5][7] = {"мяу! ", "мяяяу!", "м ", "! ", "му! "};

for (int i = 0; i < 5; i++)

{ int j = 0;

int q = 0;

cout << q << endl;

bool b = true;//q = 4

while (b && j < 7)

{ char x = s[i][j];

switch (q)

{case 0: if (x == 'м')

{ q = 1;

cout << q << endl;

}

else b = false;

break;

case 1: if (x == 'я')

{ q = 2;

cout << q << endl;}

else b = false;

break;

case 2: if (x == 'я')

{ q = 2;

cout << q << endl;}

else if (x == 'у')

{q = 3;

cout << q << endl;}

else b = false;

break;

case 3: if (x == '!')

{ q = 4;

cout << q << endl;}

else b = false;

break;

default: break;

}

j++;

}

if (q == 4) cout << "слово принадлежит языку" << endl;

else cout << "слово не принадлежит языку" << endl;

}

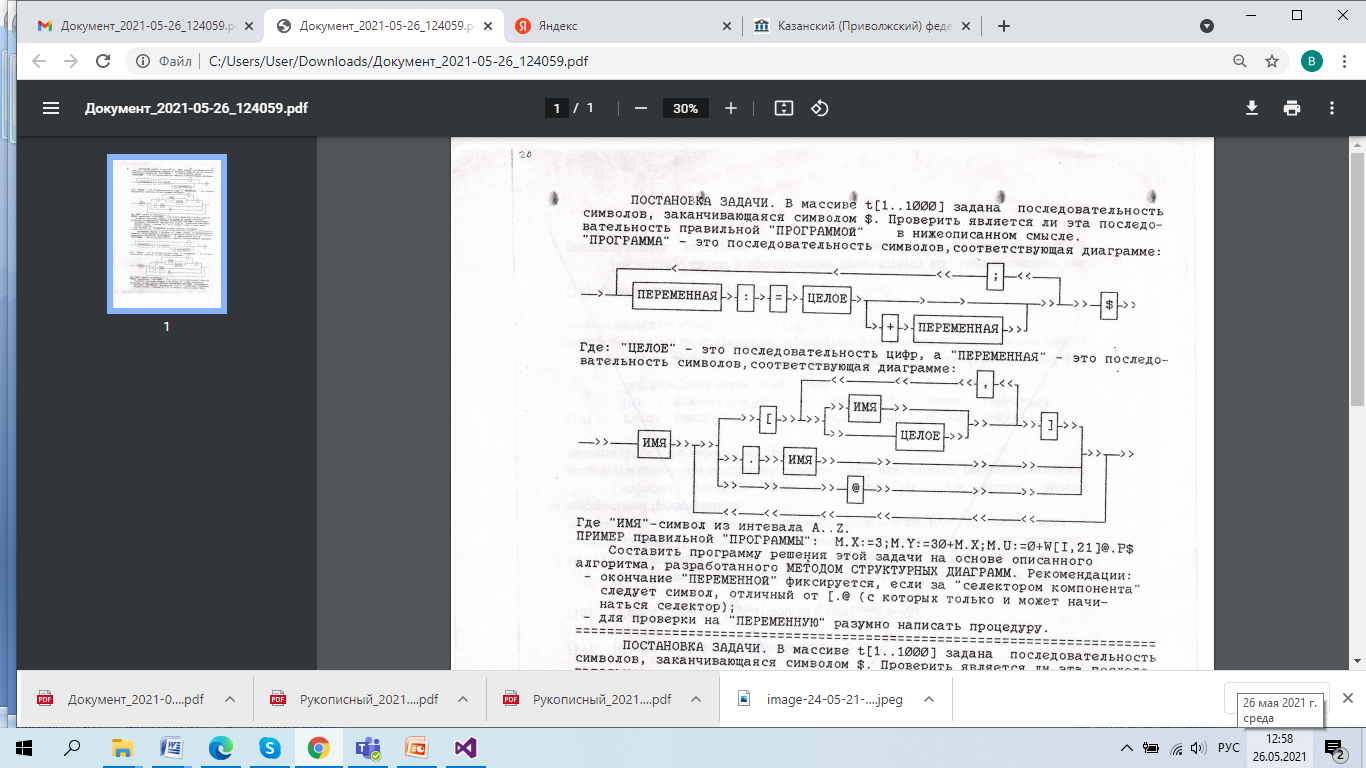
system ("pause");

return 0;

}

1. **Метод структурных диаграмм**.

В билете будет описан язык, нарисована его структурная диаграмма.



Нужно будет написать код по этой диаграмме и протестировать код.

if (!in)

{

cout << "can’t open file!";

exit(1);

}

else

{

in >> x;

while (!in.eof())

{

if (x >= 'a' && x <= 'z')

outText << x;

else if (x == '{')

do

{

in >> x;

} while (!in.eof() && x != '}');

else if (x == '\'')

{

outText << x;

while (!in.eof() && x != '\'')

{

in >> x;

outText << x;

}

}

else cout << "ошибка в тексте" << endl;

in >> x;

}

}

in.close();

outText.close();

system("pause");

return 0;

1. **Метод структурных диаграмм на файлах**

Задача «Турнир по информатике» из лекции 27. Структура будет нарисована. Нужно будет написать код и протестировать его.

void formNewBin(ifstream & fD, ifstream & fP, ofstream & gN, ofstream & gP)

{

typeStruct xDog, xPst, xPere, xNedo;

fD.read((char\*)&xDog, sizeof(typeStruct));

fP.read((char\*)&xPst, sizeof(typeStruct));

//while (!fD.eof() && !fP.eof())

while (xDog.kPst != 100 || xDog.kDet != 100 || xPst.kPst != 100 || xPst.kDet != 100)

{

if (xDog.kPst < xPst.kPst || xDog.kPst == xPst.kPst && xDog.kDet < xPst.kDet)//КлючДоговора < КлючПоставки

//обработать договор без поставок

{

//это недовыполненный договор

gN.write((char\*)&xDog, sizeof(typeStruct));

fD.read((char\*)&xDog, sizeof(typeStruct));

}

else

if (xDog.kPst > xPst.kPst || xDog.kPst == xPst.kPst && xDog.kDet > xPst.kDet) //КлючДоговора > КлючПоставки

//обработать поставку без договора

{

//это перевыполненный договор, собираем все поставки с этим ключом

xPere = xPst;

xPere.kol = 0;

do

{

xPere.kol += xPst.kol;

fP.read((char\*)&xPst, sizeof(typeStruct));

} while (!fP.eof() && xPere.kPst == xPst.kPst && xPere.kDet == xPst.kDet);

gP.write((char\*)&xPere, sizeof(typeStruct));

}

else //обработать поставку с договором

{

//обрабатываются все поставки с этим ключом

double s = 0;

do

{

s += xPst.kol;

fP.read((char\*)&xPst, sizeof(typeStruct));

} while (!fP.eof() && xDog.kPst == xPst.kPst && xDog.kDet == xPst.kDet);//!!!!

if (s < xDog.kol) //

{

xNedo = xDog;

xNedo.kol -= s;

gN.write((char\*)&xNedo, sizeof(typeStruct));

}

else if (s > xDog.kol) //

{

xPere = xDog;

xPere.kol = s - xPere.kol;

gP.write((char\*)&xPere, sizeof(typeStruct));

}

// else //договор выполнен полностью, его пропускаем

// сейчас ключ поставки не совпадает с ключом договора

//читаем новый договор

fD.read((char\*)&xDog, sizeof(typeStruct));

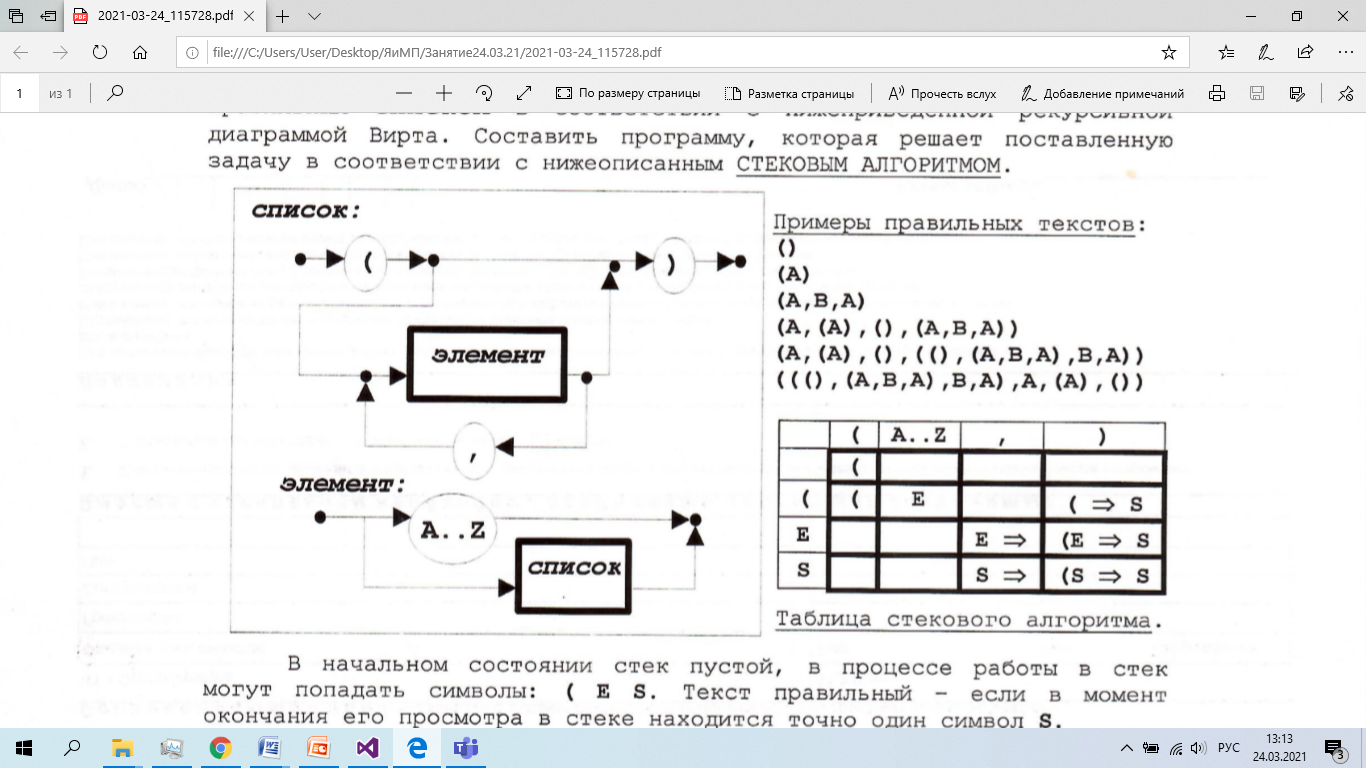
}

}

}

1. **Стековый алгоритм.**

Пример задачи на стековый алгоритм – бонусная задача.



Пример правильного списка: (a,v,(d),(e,f,d))

void main()

{

setlocale(LC\_ALL, "Russian");

char x;

char stack[100];

int top = -1;//указывает на последнюю занятую позицию

bool b = true;

ifstream f("Text.txt");

while (f.peek() != EOF && b)

{

f.get(x);

if (top == -1 || stack[top] == '(' || stack[top] == 'Z')

{

if (x >= 'a'&& x <= 'z')

{

top += 1;

stack[top] = 'E';

}

else

if (x == '(')

{

top += 1;

stack[top] = '(';

}

else b = false;

}

else // stack[top] = 'E'

{

if (x == '+' || x == '-' || x == '\*' || x == '/')

{

top += 1;

stack[top] = 'Z';

}

else if (x == ')')

{

if (top >= 3 && stack[top - 1] == 'Z' && stack[top - 2] == 'E' && stack[top - 3] == '(')

{

top -= 3;

stack[top] = 'E';

}

else b = false;

}

//else b = false;

}

}

if (b && top == 0 && stack[top] == 'E')

cout << "Текст правильный " << endl;

else cout << "Текст неправильный " << endl;

system("pause");

}

1. **Списки**

Всевозможные задачи на списки.

- Например, задачи на редактирование текста (файл у вас был)

- Просто работа со списками. Например:

Включить в список ***l3*** те компоненты из списка ***l1*** , для которых в списке ***l2*** найдется хотя бы одна компонента, меньшая него. Списки ***l1*** и ***l2*** не пустые и не упорядоченные.

- Некоторые задачи нужно будет оформить как многофайловый проект.

#include <iostream>

#include <fstream>

using namespace std;

struct node

{

int info;

node\* next;

};

void createListFromFileByTail(node\*& head1, ifstream& f1)

{

node\* q;

if (f1.peek() == NULL)

{

head1 = NULL;

cout << "++++++++++ File is empty" << endl;

}

else

{

head1 = new node;

head1->next = NULL;

node\* tail = head1;

while (!f1.eof())

{

q = new node;

f1 >> q->info;

q->next = NULL;

tail->next = q;

tail = q;

}

q = head1;

head1 = head1->next;

delete q;

}

}

void showList(node\*& head1)

{

if (head1 == NULL) cout << "spisok pust" << endl;

else

{

node\* q = head1;

while (q != NULL)

{

cout << q->info << ' ';

q = q->next;

}

cout << endl;

}

}

void deleteList(node\*& head1)

{

node\* p = head1;

while (head1 != NULL)

{

head1 = head1->next;

delete p;

p = head1;

}

}

ifstream f1("input1.txt"), f2("input2.txt");

node\* L1 = nullptr, \* L2 = nullptr, \* L = nullptr;

createListFromFileByTail(L1, f1);

createListFromFileByTail(L2, f2);

struct nodeDL //nodeDoubleList

{

int data;

nodeDL \* next;

nodeDL \* prev;

};

* проверка двусвязного списка на пустоту

//проверка двусвязного списка на пустоту

bool isEmptyDL(nodeDL \* head)

{

return head == NULL;

}

* подсчет длины двусвязного списка

// подсчет длины двусвязного списка

int lengthDL(nodeDL \* head)

{

nodeDL \* p;

int len;

if (isEmptyDL(head)) len = 0;

else

{

len = 0;

p = head;

while (p)

{

p = p->next;

len++;

}

}

return len;

}

* создание двусвязного списка

//создание двусвязного списка добавлением элемента в конец списка

void createDL(nodeDL\* & head, nodeDL\* & end, ifstream & f)

{

nodeDL\* p;

head = NULL;

end = NULL;

while (!f.eof())

{

p = new nodeDL;

f >> p->data;

if (head == NULL)

{

head = p;

p->next = NULL;

p->prev = NULL;

end = p;

}

else

{

end->next = p;

p->prev = end;

p->next = NULL;

end = p;

}

}

}

* печать двусвязного списка

//печать двусвязного списка

void showDL(nodeDL \* head)

{

nodeDL \* p = head;

while (p)

{

cout << p->data << ' ';

p = p->next;

}

cout << endl;

}

8. **Деревья**

Задачи на обходы дерева с попутным поиском ответа на вопрос. При этом мы в основном будем строить деревья поиска, но не будем пользоваться его свойством упорядоченности при обходах. Закроем на это глаза.

- Построить список из положительных элементов заданного бинарного дерева

- Создать копию заданного дерева

- Дано дерево поиска. Сместить его корень на одну вершину вправо так, чтобы снова получилось дерево поиска (здесь как раз учитываем, что это именно дерево поиска)

Возможно, некоторые задачи нужно будет оформить как многофайловый проект, с выделением стека, очереди и т.п.

struct treeNode

{

treeNode\* left;

treeNode\* right;

int data;

};

//опишем структуру для очереди

struct queueNode

{

queueNode\* next;

treeNode\* node;

};

//опишем операции работы с стеком

//поставить элемент в очередь - в конец списка

void push\_queue(queueNode\* &first, queueNode \*&last, treeNode\* t)

{

queueNode\* p;

if (first)

{

p = new queueNode;

p->node = t;

p->next = NULL;

last->next = p;

last = p;

}

else

{

p = new queueNode;

p->node = t;

p->next = NULL;

first = p;

last = p;

}

}

//взять элемент из очереди (удалить первый элемент из списка)

void pop\_queue(queueNode\* &first, treeNode\* &t)

{

queueNode \*p;

t = first->node;

p = first->next;

delete first;

first = p;

}

//печать содержимого узла

void print\_node(int x)

{

cout << " " << x;

}

//собственно обход дерева в глубину

void obchod(treeNode\* root)

{

queueNode \*first, \*last;

treeNode \*p;

p = root;

if (p)

{

first = NULL;

last = NULL;

push\_queue(first, last, p);

do

{

pop\_queue(first, p);

print\_node(p->data);

if (p->left)

push\_queue(first, last, p->left);

if (p->right)

push\_queue(first, last, p->right);

} while (first);

}

}

//опишем функцию создания вершины дерева

treeNode\* create\_node(int x)

{

treeNode\* m = new treeNode;

m->data = x;

m->left = NULL;

m->right = NULL;

return m;

}

//поместить новую вершину в дерево поиска

void push\_tree(treeNode\* &root, int x)

{

treeNode \*pred = root, \*marker;

bool found = false;

if (root == NULL)

{

root = create\_node(x);

}

else

{

marker = root;

while (!found && marker != NULL)

{

pred = marker;

if (marker->data == x) found = true;

else

if (x >= marker->data)

marker = marker->right;

else marker = marker->left;

}

if (!found)

if (x >= pred->data)

pred->right = create\_node(x);

else

pred->left = create\_node(x);

}

}

// Создание дерева поиска

void create\_tree(ifstream&f, treeNode\* &root)

{

root = NULL;

int x;

while (f >> x)

{

push\_tree(root, x);

}

}

//печать дерева

void showTree(treeNode\* root)

{

if (root != NULL)

{

showTree(root->left);

cout << root->data << ' ';

showTree(root->right);

}

}