Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

**Национальный исследовательский университет «МЭИ»**

# Типовой расчет «Моделирование типов. Ссылочные структуры»

**Вариант 21**

*Задание выполнил: Челышев Э. А.*

*Студент группы A-8-17*

*Проверил: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Оценка:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Замечания:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Москва 2018 НИУ «МЭИ»

**Часть 1**

Задание:

**Задание В – «БИНАРНОЕ ДЕРЕВО»**

**Часть В1. Смоделировать двумя способами** новый тип «Бинарное дерево»:

1. на основе ссылочного типа «Нелинейный связный список»;
2. на основе динамического массива или типизированного/двоичного файла (на

выбор студента).

Для этого **создать модуль** и описать **новый тип данных** и его структуру (на

русском языке и на Delphi/C, без классов! Только записи/структуры и массив/файл)

и **базовые операции**:

- проверка на пустоту дерева (поддерева, начинающегося с текущей вершины);

- создание пустого дерева или дерева из одного элемента (как удобнее);

- сделать текущим (по адресу в списке или индексу в массиве) корень дерева;

- сделать текущим левое поддерево (от текущей вершины);

- сделать текущим правое поддерево (от текущей вершины);

- сделать текущим родительскую вершину для текущей вершины (для удобства

можно в способе 1 ввести третью связь – обратную связь с родительской вершиной,

чтобы не искать ее по всему дереву – Трѐхсвязный список);

- узнать значение текущей вершины;

- изменить значение текущей вершины;

- создать левое поддерево для текущей вершины;

- создать правое поддерево для текущей вершины;

**Первый способ: «Нелинейный двусвязный список»**

Каждый элемент дерева представляется структурой TTree:

struct TTree {

TInfo Info;

TTree \*left;

TTree \*right;

TTree \*high;};

Где: тип TInfo в свою очередь является структурой, включающей в себя структуру date типа Tdate:

struct TInfo{

char F[20];

char I[20];

char O[20];

struct Tdate{

int de;

int m ;

int g ;

}date;};

\*left – указатель на левое поддерево (равен NULL при отсутствии)

\*right – указатель на правое поддерево (равен NULL при отсутствии)

\*high - указатель на родительскую вершину (равен NULL при отсутствии)

Само дерево представляется структурой Tder (конкретно переменная d данного типа) :

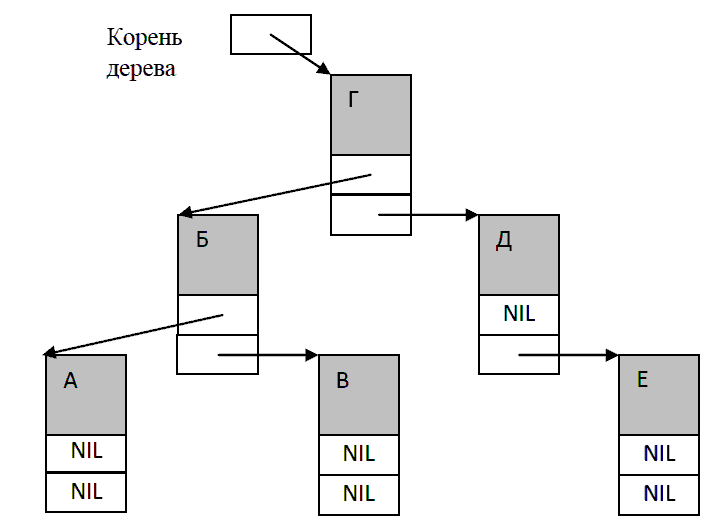
struct Tder{

TTree \*root, \*tek; };

\*root – указатель на корень дерева.

\*tek – указатель на текущий элемент дерева.

Схема:



**Базовые операции**

1. **проверка на пустоту дерева (поддерева, начинающегося с текущей вершины);**

int IsEmpty (Tder d, TTree \*p) {

if (p==NULL || strcmp(p->Info.F,"\*\*")==0) return 1;

else return 0;}

Функция возвращает 1, если текущий элемент пуст, в противном случае 0.

1. **создание пустого дерева или дерева из одного элемента (как удобнее);**

void Create1( Tder &d, TInfo info) //

{d.root = new TTree;

strcpy((d.root)->Info.F, info.F);

strcpy((d.root)->Info.I, info.I);

strcpy((d.root)->Info.O, info.O);

(d.root)->Info.date.de=info.date.de;

(d.root)->Info.date.m=info.date.m;

(d.root)->Info.date.g=info.date.g;

(d.root)->left=NULL;

(d.root)->right=NULL;

d.tek=d.root;

(d.tek)->high=d.root;}

При этом указатели на левое и правое поддерево объявляются равными NULL, а указатель на родительскую вершину равен указателю на текущий элемент, то есть дерево из одного элемента как бы зацикливается на себе.

1. **сделать текущим (по адресу в списке или индексу в массиве) корень дерева;**

void GetRoot(Tder &d){

if (d.tek) d.tek=d.root;}

Указатель на текущий элемент становится равным указателю на корень дерева.

1. **сделать текущим левое поддерево (от текущей вершины);**

void GetLeft(Tder &d){

if (d.tek) d.tek=(d.tek)->left;}

Указатель на текущий элемент становится равным указателю на левое поддерево.

1. **сделать текущим правое поддерево (от текущей вершины);**

void GetRight(Tder &d){

if (d.tek) d.tek=(d.tek)->right;}

Указатель на текущий элемент становится равным указателю на правое поддерево.

1. **сделать текущим родительскую вершину для текущей вершины;**

void GetHigh(Tder &d){

if (d.tek) d.tek=(d.tek)->high;}

Использование указателя на родительскую вершину позволяет осуществить переход в одно элементарное действие, в противном случае нам пришлось бы искать родительскую вершину по всему дереву.

1. **узнать значение текущей вершины;**

TInfo GetInfo(Tder &d){ //получить значение текущей вершины

if (d.tek) return (d.tek)->Info;}

Функция возвращает значение типа TInfo, равное полю Info структуры TTree.

1. **изменить значение текущей вершины;**

void Change(Tder &d,TInfo info){

(d.tek)->Info=info;}

Функция изменяет значение поля Info.

1. **создать левое поддерево для текущей вершины;**

void AddLeft(Tder d){

d.tek->left= new TTree;

(d.tek)->left->high=d.tek;

(d.tek)->left->right=NULL;

(d.tek)->left->left=NULL;

strcpy((d.tek)->left->Info.F, "\*\*");

strcpy((d.tek)->left->Info.I, "\*\*");}

Функция выделяет память для левого поддерева, указателю на родительскую вершину в поддереве присваивается значение указателя на текущую вершину. Указатели \*left и \*right в добавленном элементе равны NULL, сам элемент пуст.

1. **создать правое поддерево для текущей вершины;**

void AddRight(Tder d){

d.tek->right= new TTree;

(d.tek)->right->high=d.tek;

(d.tek)->right->right=NULL;

(d.tek)->right->left=NULL;

strcpy((d.tek)->right->Info.F, "\*\*");

strcpy((d.tek)->right->Info.I, "\*\*");

}

Функция выделяет память для правого поддерева, указателю на родительскую вершину в поддереве присваивается значение указателя на текущую вершину. Указатели \*left и \*right в добавленном элементе равны NULL, сам элемент пуст.

1. **удаление дерева;**

void free\_tree(Tder &d){//а d.root тоже ослободить надо

rec(d);

d.root=NULL;}

Единственная из базовых операций, определенных с использованием рекурсии.

Функция вызывает подфункцию rec(d), которая, обходя дерево в ширину слева направо, очищает память и удаляет элементы дерева из памяти, а затем присваивает указателю на корень дерева значение NULL.

void rec( Tder &d){

if (!IsEmpty(d, d.tek))

{ GetLeft(d);

rec(d);

GetHigh(d);

GetRight(d);

rec(d);

GetHigh(d);

delete d.tek;

d.tek=NULL;

}}

1. **Вывод в текстовый лог-файл.**

Кроме стандартных операций, используемых для создания и обработки дерева, есть так же операция вывода дерева в текстовый лог-файл для контроля за текущим состоянием дерева после выполнения любой операции из части 2. Данная задача осуществляется непосредственно отдельными операциями части 2, то есть вывод просто дублируется, одинаковые результаты мы видим и в консоли, и лог-файле. Если выполнение операции из части 2 не дает представления о структуре дерева, то используется следующая функция вывода дерева в ширину (по сути своей она выводит дерево в том же представлении, что и во входном файле):

void print\_file (FILE \*log, Tder d){

TInfo info;

GetRoot;

if (!IsEmpty(d, d.tek)) {

info=GetInfo(d);

fprintf(log, "%s %s %s, род. %2d.%2d.%4d\n", info.F, info.I,info.O,info.date.de, info.date.m, info.date.g);

GetLeft(d);

print\_file (log, d);

GetHigh(d);

GetRight(d);

print\_file (log, d);

GetHigh(d);}

else fprintf(log,"\*\*\n");}

Функция изначально проверяет текущий элемент на пустоту, в случае, если он не пуст, выводит информацию, содержащуюся в поле Info, в противном случае выводит «\*\*».

**Второй способ: «Бинарный файл»**

Дерево представляется структурой Tder:

struct Tder{

FILE \*fb;

char \* name;

int root, tek;

};

\*fb – указатель на бинарный файл

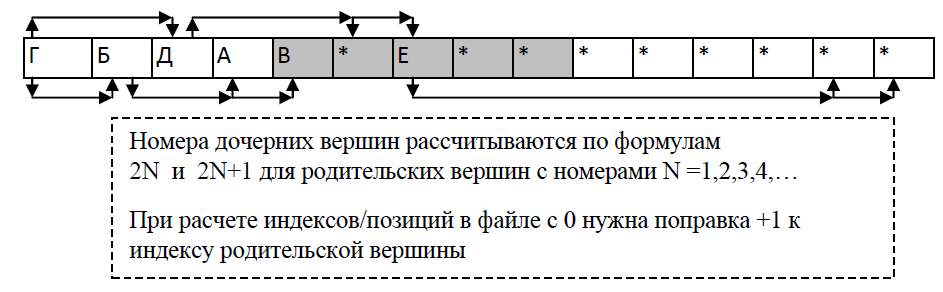
name – имя бинарного файла

root – индекс корня дерева.

tek – индекс текущего элемента.

Кроме того, в программе для представления данных используется следующая структура:

**Схема**

****

**Базовые операции**

1. **проверка на пустоту дерева (поддерева, начинающегося с текущей вершины);**

int IsEmpty (Tder d, int p) {

TInfo info;

int n;

if (d.fb==NULL) return 1;

fseek(d.fb, p\*sizeof(TInfo),SEEK\_SET);

n=fread(&info, sizeof(TInfo), 1, d.fb);

if (n==0 || strcmp(info.F,"\*\*")==0) {

fseek(d.fb, p\*sizeof(TInfo),SEEK\_SET);return 1;} else {

fseek(d.fb, p\*sizeof(TInfo),SEEK\_SET); return 0;}

}

Функция возвращает 1, если текущий элемент пуст (или его просто не удалось считать), в противном случае 0.

1. **создание пустого дерева или дерева из одного элемента (как удобнее);**

void Create1(Tder &d, TInfo info) //

{ TInfo inf;

strcpy(inf.F, "\*\*");

strcpy(inf.I, "\*\*");

strcpy(inf.O, "\*\*");

inf.date.de=0;

inf.date.m=0;

inf.date.g=0;

d.name="derevo";

d.fb=fopen(d.name, "wb+");

for (int i=0; i<8; i++) fwrite(&inf, sizeof(TInfo), 1, d.fb);

d.root = 1;

d.tek=d.root;

fseek(d.fb, 1\*sizeof(TInfo),SEEK\_SET);

fwrite(&info, sizeof(TInfo), 1, d.fb);

fseek(d.fb, 1\*sizeof(TInfo),SEEK\_SET);

}

Функция создает бинарный файл, открывая его для чтения и записи. Затем создаются 8 пустых записей про запас, а в первую (не считая нулевую) помещается переданное функции значение info типа TInfo.

1. **сделать текущим (по адресу в списке или индексу в массиве) корень дерева;**

void GetRoot(Tder &d){

if (d.tek) {d.tek=d.root; fseek(d.fb, d.root\*sizeof(TInfo), SEEK\_SET);}

}

Индекс текущего элемента меняется, переход к нужной позиции в файле.

1. **сделать текущим левое поддерево (от текущей вершины);**

void GetLeft(Tder &d){ //сделать текущим левое поддерево

if (d.tek) d.tek\*=2;

fseek(d.fb, d.tek\*sizeof(TInfo),SEEK\_SET);

}

Индекс текущего элемента меняется, переход к нужной позиции в файле.

1. **сделать текущим правое поддерево (от текущей вершины);**

void GetRight(Tder &d){//сделать текущим правое поддерево

if (d.tek) d.tek=d.tek\*2+1;

fseek(d.fb, d.tek\*sizeof(TInfo),SEEK\_SET);

}

Индекс текущего элемента меняется, переход к нужной позиции в файле.

1. **сделать текущим родительскую вершину для текущей вершины;**

void GetHigh(Tder &d){//сделать текущим правое поддерево

d.tek=d.tek/2;

if (d.tek) fseek(d.fb, d.tek\*sizeof(TInfo),SEEK\_SET);

}

Индекс текущего элемента меняется, переход к нужной позиции в файле.

1. **узнать значение текущей вершины;**

TInfo GetInfo(Tder &d){

TInfo info;//получить значение текущей вершины

if (d.tek){ fread(&info, sizeof(TInfo), 1, d.fb);

fseek(d.fb, d.tek\*sizeof(TInfo), SEEK\_SET); return info;}

}

Функция возвращает значение типа TInfo, расположенное в бинарном файле на позиции, равной индексу текущего элемента, умноженной на размер типа.

1. **изменить значение текущей вершины;**

void Change(Tder &d,TInfo info){

fseek(d.fb, d.tek\*sizeof(TInfo), SEEK\_SET);

fwrite(&info, sizeof(TInfo), 1, d.fb);

fseek(d.fb, d.tek\*sizeof(TInfo), SEEK\_SET);

}

Функция изменяет значение, расположенное на заданной позиции в файле.

1. **создать левое поддерево для текущей вершины;**

void AddLeft(Tder d){

TInfo inf;

strcpy(inf.F, "\*\*");

strcpy(inf.I, "\*\*");

strcpy(inf.O, "\*\*");

inf.date.de=0;

inf.date.m=0;

inf.date.g=0;

fseek(d.fb, d.root\*sizeof(TInfo), SEEK\_END);

if(ftell(d.fb)<d.tek\*4)

for(int i=1; i<=pow(2, (int)sqrt(d.tek)+2); i++)

fwrite(&inf, sizeof(TInfo), 1, d.fb);

fseek(d.fb, d.tek\*sizeof(TInfo), SEEK\_SET);

}

1. **создать правое поддерево для текущей вершины;**

void AddRight(Tder d){

TInfo inf;

strcpy(inf.F, "\*\*");

strcpy(inf.I, "\*\*");

strcpy(inf.O, "\*\*");

inf.date.de=0;

inf.date.m=0;

inf.date.g=0;

fseek(d.fb, 0, SEEK\_END);

if(ftell(d.fb)<((d.tek\*2+1)\*2)\*sizeof(TInfo))

for(int i=1; i<=(int)sqrt(d.tek)+2; i++)

fwrite(&inf, sizeof(TInfo), 1, d.fb);

fseek(d.fb, d.tek\*sizeof(TInfo), SEEK\_SET);

}

Функции AddLeft и AddRight фактически сводятся к проверке наличия места для нового элемента и его левого и правого поддеревьев. В случае, если пустых записей в файле для них не хватает, функция создает в бинарном файле целых два яруса: для элементов дерева, находящихся на одном уровне с добавляемыми элементами, и их дочерних элементов.

1. **удаление дерева;**

void free\_tree(Tder &d){ //при освобождении памяти сообщать

fclose(d.fb);

remove(d.name);

}

Функция сводится к удалению файла.

1. **Вывод в текстовый лог-файл.**

Остается неизменным, так как функция написана с использованием исключительно стандартных операций над деревом, без прямого обращения к указателям либо бинарному файлу.

**Часть 2**

**Создать приложение** (консольное, на C или Delphi), позволяющее

выбрать и выполнить следующие действия над деревом (только с помощью выше

перечисленных базовых операций! Без прямого обращения к элементам

списка/массива/файла):

- удалить дерево (освободить память);

- создать упорядоченное по ключу (по возрастанию/убыванию одного или

нескольких полей записи/структуры) бинарное дерево из текстового файла;

- создать неупорядоченное бинарное дерево (генеалогическое дерево) из текстового

файла (Корень – Левое\_поддерево – Правое\_поддерево, \*\* – признак пустого

поддерева);

- добавить один элемент, со значениями введенными в интерактивном режиме (с

клавиатуры) в упорядоченное дерево по ключу (по возрастанию/убыванию одного

или нескольких полей записи/структуры) для построения упорядоченного дерева;

например, в ниже изображенное дерево добавляются числа 6,3,4,2,6,9 по

возрастанию (неубыванию в случае добавления одинаковых значений);

- найти решение задачи из списка заданий по вариантам в конце этого файла

(стр.16);

- вывести дерево 2-4мя разными способами: (Корень – Левое\_поддерево –

Правое\_поддерево, Левое\_поддерево – Корень – Правое\_поддерево,

Левое\_поддерево – Правое\_поддерево – Корень; графически(по желанию). \*\* –

признак пустого поддерева);

21. а) Создав соответствующее упорядоченное дерево, вывести информацию обо

всех людях в алфавитном порядке по именам, при совпадении имен – по фамилиям.

б) В генеалогическом дереве подсчитать количество родственников, родившихся в

указанный временной период (задается в интерактивном режиме).

**Формат входного текстового файла:**

<Фамилия>

<Имя>

<Отчество>

<День рождения>

<Месяц рождения>

<Год рождения>

…

<Фамилия>

<Имя>

<Отчество>

<День рождения>

<Месяц рождения>

<Год рождения>

…

\*\*

…

**Интерфейс приложения**

N - создать упорядоченное по ключу дерево

G - создать генеалог. дерево

V - вывести дерево в ширину

W - вывести дерево в глубину

F - Освободить дерево

A - добавить элемент в дерево

D - найти решение

E – выход.

**Код функций:**

1. **создать упорядоченное по ключу дерево**

while(!feof(f)){

inf=vvod(f);

if (!feof(f) && strcmp(inf.F,"\*\*")){ sort(d,inf);

c++; }

GetRoot(d);

}

**Функция vvod:**

TInfo vvod(FILE \*f)

{TInfo info;

fscanf(f, "%s", info.F);

if (strcmp(info.F,"\*\*")){

fscanf(f, "%s", info.I);

fscanf(f, "%s", info.O);

fscanf(f,"%2d", &info.date.de);

fscanf(f,"%2d", &info.date.m);

fscanf(f,"%4d", &info.date.g);

}

return info;

}

**Функция sort:**

void sort(Tder &d, TInfo info){

char\* str="АаБбВвГгДдЕеЁёЖжЗзИиЙйКкЛлМмНнОоПпРрСсТтУуФфХхЦцЧчШшЩщЪъЫыЬьЭэЮюЯя";

int i=0, dlin;

int flag=1;

TInfo inf=GetInfo(d);;

if (IsEmpty(d, d.root)){

Create1(d, info);

Change(d, info);

AddRight(d);

AddLeft(d);}

else {

if (!IsEmpty(d, d.tek)){

if (strlen(info.I)<strlen(inf.I)) dlin=strlen(info.I); else dlin =strlen(inf.I);

while(flag==1 && i<=dlin){

if ((strchr(str, info.I[i])-str)/2 < (strchr(str, inf.I[i])-str)/2)

flag=0;

if ((strchr(str, info.I[i])-str)/2 > (strchr(str, inf.I[i])-str)/2)

flag=2;

i++;

}

if (flag==1 && strlen(info.I)>strlen(inf.I)) i=2;

if (flag==1 && strlen(info.I)<strlen(inf.I)) i=0;

if (flag==1) {

if (strlen(info.F)<strlen(inf.F)) dlin=strlen(info.F); else dlin =strlen(inf.F);

i=0;

while(flag==1 && i<=dlin){

if ((strchr(str, info.F[i])-str)/2 < (strchr(str, inf.F[i])-str)/2)

flag=0;

if ((strchr(str, info.F[i])-str)/2 > (strchr(str, inf.F[i])-str)/2)

flag=2;

i++;

}

if (flag==1 && strlen(info.F)>strlen(inf.F)) i=2;

if (flag==1 && strlen(info.F)<strlen(inf.F)) i=0;

}

if (flag==1) flag=0;

switch (flag) {

case 0:

{ GetLeft(d);

sort(d, info);

break;

}

case 2:

{ GetRight(d);

sort(d, info);

break;

}

}

}

else {Change(d,info);

AddRight(d);

AddLeft(d);}

} }

1. **создать генеалогическое дерево**

void create\_tree(Tder &d, FILE \*f, int &kol){

TInfo info;

fscanf(f, "%s", info.F);

if (feof(f)) return;

if (strcmp(info.F,"\*\*")){

fscanf(f, "%s", info.I);

fscanf(f, "%s", info.O);

fscanf(f,"%2d", &info.date.de);

fscanf(f,"%2d", &info.date.m);

fscanf(f,"%4d", &info.date.g);

if (IsEmpty(d, d.root))

Create1(d, info);

Change(d, info);

AddLeft(d);

GetLeft(d);

create\_tree(d, f, kol);

GetHigh(d);//

AddRight(d);

GetRight(d);

create\_tree(d, f, kol);

GetHigh(d);

kol++;

}}

1. **вывести дерево в ширину**

void vivod (Tder d, int i, FILE \*log){

TInfo info;

if (!IsEmpty(d, d.tek)) {

info=GetInfo(d);

GetLeft(d);

vivod (d, i+1, log);

GetHigh(d);

if (i==0) {

printf("%s %s %s, род. %2d.%2d.%4d\n", info.F, info.I,info.O,info.date.de, info.date.m, info.date.g);

fprintf(log, "%s %s %s, род. %2d.%2d.%4d\n", info.F, info.I,info.O,info.date.de, info.date.m, info.date.g);}

else {

printf("%\*s %s %s %s, род. %2d.%2d.%4d\n", i\*8, "--------", info.F, info.I,info.O,info.date.de, info.date.m, info.date.g);

fprintf(log, "%\*s %s %s %s, род. %2d.%2d.%4d\n", i\*8, "--------", info.F, info.I,info.O,info.date.de, info.date.m, info.date.g);

}

GetRight(d);

vivod (d, i+1, log);

GetHigh(d);

}}

1. **вывести дерево в глубину**

void make\_tree(Tder &d, FILE \*f, int i, FILE \*log){

TInfo info;

info=GetInfo(d);

if (!IsEmpty(d, d.tek)) {

if (i==0) {printf("%s %s %s, род. %2d.%2d.%4d\n", info.F, info.I,info.O,info.date.de, info.date.m, info.date.g);

fprintf(log, "%s %s %s, род. %2d.%2d.%4d\n", info.F, info.I,info.O,info.date.de, info.date.m, info.date.g);}

else{ printf("%\*s %s %s %s, род. %2d.%2d.%4d\n", i\*8, "--------", info.F, info.I,info.O,info.date.de, info.date.m, info.date.g);

fprintf(log, "%\*s %s %s %s, род. %2d.%2d.%4d\n", i\*8, "--------", info.F, info.I,info.O,info.date.de, info.date.m, info.date.g);}

GetLeft(d);

make\_tree(d, f, i+1, log);

GetHigh(d);

GetRight(d);

make\_tree(d, f, i+1, log);

GetHigh(d);

}

else {printf("%\*s", i\*8, "--------\n");

fprintf(log, "%\*s", i\*8, "--------\n");}}

1. **Освободить дерево**

См. стандартные

1. **добавить элемент в дерево**

Используется процедура **sort**, описанная в п.1

fprintf(log, "\nДобавление элемента:\n");

TInfo info;

printf("Фамилия\n");

scanf("%s", info.F);

printf("Имя\n");

scanf("%s", info.I);

printf("Отчество\n");

scanf("%s", info.O);

printf("Дата\n");

scanf("%2d.%2d.%4d", &info.date.de, &info.date.m, &info.date.g);

fflush(stdin);

sort(d, info);

printf("Элемент успешно добавлен\n");

GetRoot(d);

fprintf(log, "Элемент '%s %s %s, род. %2d.%2d.%4d' успешно добавлен\n\n", info.F, info.I, info.O, info.date.de,info.date.m,info.date.g);

print\_file(log, d);

fprintf(log,"========================================================================\n\n");

GetRoot(d);

1. **найти решение**

int Decide(Tder d, int d1, int d2, int m1, int m2, int g1, int g2){

int count=0;

TInfo info=GetInfo(d);

if (!IsEmpty(d, d.tek)){

if (info.date.g<g2 && info.date.g>g1) count++;

if ((info.date.g==g2 || info.date.g==g1) && info.date.m<m2 && info.date.m>m1) count++;

if ((info.date.g==g2 || info.date.g==g1) && (info.date.m==m2 || info.date.m==m1) && info.date.de<=d2 && info.date.de>=d1) count++;//printf("%s %s %s, ???. %2d.%2d.%4d", info.F, info.I,info.O,info.date.de, info.date.m, info.date.g);

GetLeft(d);

count+=Decide(d, d1, d2, m1, m2, g1, g2);

GetHigh(d);//

GetRight(d);

count+=Decide(d, d1, d2, m1, m2, g1, g2);

GetHigh(d);

return count;

} else return 0;

}

**Функциональные тесты**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № теста | Входные данные | Применяемые операции | Ожидаемый результат | Смысл теста |
| 1 | Пустой текстовый файл | Создание упорядоченного дерева или  Создание генеалогического дерева | Не удалось создать дерево | Случай с пустым файлом |
| 2 | ---- | 2.Вывод дерева в ширину или  Вывод дерева в высоту  *При условии, что дерево пусто* | Пустое дерево | Случай вывода пустого дерева |
| 3 | \*\* | Создание упорядоченного дерева или  Создание генеалогического дерева | Не удалось создать дерево | Случай с файлом, который по сути своей пустым не является, но не содержит ни одного элемента |
| 4 | Иванов  Петр  Иванович  11.12.1965 | Добавление элемента  *При условии, что дерево пусто* | Элемент успешно добавлен | Добавление элемента в пустое дерево |
| 5 | См. входной файл | Создание упорядоченного дерева | Дерево из 10 элементов успешно создано | Успешное создание дерева |
| 6 | См. входной файл | Создание генеалогического дерева | Генеалогическое дерево успешно создано | Успешное создание дерева |
| 7 | 11.11.1956 11.11.1998 | Нахождение решения,  *Дерево из теста 6* | Количество элементов, удовлетворяющих запросу: 4 | Люди, родившиеся в этот временной период, найдены |
| 8 | 09.08.1865 11.10.1910 | Нахождение решения  *Дерево из теста 6* | Количество элементов, удовлетворяющих запросу: 0 | Люди, родившиеся в этот временной период, не найдены |
| 9 | Иванов  Петр  Иванович  11.12.1965 | Добавление элемента  *Дерево из теста 6* | Элемент успешно добавлен | Добавление элемента в дерево. При этом из лог-файла видно, что добавлен правильно |
| 10 | Арсеньев  Максим  Максимович  16.09.1934 | Добавление элемента  *Дерево из теста 9* | Элемент успешно добавлен | Добавление элемента в дерево. При этом из лог-файла видно, что добавлен правильно |

**Заключение**

Смоделирован новый тип «Матрица» двумя способами: 1) на основе нелинейного двусвязного списка и 2) на основе бинарного файла.

1. С точки зрения удобства проектирования мне показалось, что использовать метод с нелинейного списка несколько легче, нежели бинарный файл. В бинарном файле для перехода к другому элементу приходится постоянно перемещаться по файлу, и только затем возможно исследовать этот элемент или узнать его значение. После чтения или записи позиция в файле перемещается на следующую, а не остается на текущем месте. Поэтому приходится постоянно следить за тем местом, «где мы находимся в файле». Реализация через указатели мне показалась удобнее и понятнее.
2. Реализация с использованием бинарного файла требует от нас некоторого запаса места в файле. Это, казалось бы, минус, так как расходуется лишняя память. Однако мы можем себе это позволить, так как файл хранится в долговременной памяти, которая намного больше оперативной. За счет этого на основе бинарного файла возможно создать гораздо большее бинарное дерево, нежели через указатели. Кроме того, способ с использованием указателей тоже не бесплатный, так как для каждого элемента нужно три указателя не считая содержащейся в нем информации. Каждый указатель весит 4 байта. Если предположить, что у нас в дереве 10 элементов, то только на указатели уйдет 40 байтов.
3. Однако долговременная память намного медленнее оперативной, в силу этого способ с использованием указателей будет быстрее.
4. Очистить память в бинарном файле проще, так как это делается с помощью простого удаления бинарного файла. В способе через нелинейный список, требуется применить операцию delete к каждому элементу дерева, а затем удалять указатель на корень дерева. Учитывая, что функция удаления в первой реализации определена через рекурсию, разность в скорости и количестве требуемой памяти является существенной и говорит в пользу бинарного файла.
5. Возьмём для примера дерево из 10 элементов.

**Бинарный файл:**

**32 записи (5 ярусов)**

Размер бинарного файла – 84\*32 = 2304 байт

Размер структуры Tder – 16 байт

Итого 2320 байт

**Нелинейный двусвязный список:**

Размер каждого элемента в матрице – 84 байта (три строки по 20 символов, 3 целых четырехбайтовых числа и 3 указателя)

Размер структуры типа Tder – 8 байт (2 указателя)

В итоге – 84\*10 + 8 = 848 байт

Таким образом, получается, что чем больше дерево, тем дороже нам обойдется реализация через бинарный файл. Но следует учитывать, что речь идет о различных видах памяти и 2320 байт для долговременной памяти в относительном измерении меньше, чем 848 для оперативной.

**Вывод:** И так, обе реализации имеют свои плюсы и минусы. Нелинейный список является более быстрым способом, однако использует оперативную память и ограничен по объему. А файловая реализация хотя требует больше места, хранится в долговременной памяти, за счет чего мы можем создавать большие деревья. А вот скорость такого способа значительно меньше. Но лично для меня, способ с использованием нелинейного двусвязного списка оказался более интересен в описании, показался более гибким.

**Приложения**

1. Программный код приложения:

#pragma hdrstop

#pragma argsused

#include "Unit1.h"

#include <stdio.h>

#include <windows.h>

#include <conio.h>

void vivod (Tder d, int i, FILE \*log);

void make\_tree(Tder &d, FILE \*f, int i, FILE \*log);

void create\_tree(Tder &d, FILE \*f, int &kol);

void make\_tree(Tder &d, int i);

TInfo vvod(FILE \*f);

int Decide(Tder d, int d1, int d2, int m1, int m2, int g1, int g2);

void sort(Tder &d, TInfo info);

void print\_file (FILE \*log, Tder d);

Tder d;

int main(int argc, char\* argv[]) {

FILE \*f, \*log=fopen(argv[2],"w");

TInfo inf;

char ch;

int kol=0;

SetConsoleOutputCP(1251);

SetConsoleCP(1251);

do{

printf\nN – Создать упорядоченное по ключу дерево\nG – создать генеалог. дерево\nV – вывести дерево в ширину\nW – вывести дерево в глубину\nF – Освободить дерево\nA – Добавить элемент в дерево\nD – найти решение\nE - выход. \n");

ch=getchar();

printf("\n");

fflush(stdin);

ch=toupper(ch);

switch (ch) {

//----------

case 'N': {f=fopen(argv[1],"r");

int c=0;

if (f){

while(!feof(f)){

inf=vvod(f);

if (!feof(f) && strcmp(inf.F,"\*\*")){ sort(d,inf);

c++; }

GetRoot(d);

}

fclose(f);

GetRoot(d);

if (c) {printf("Дерево из %d элементов создано\n\n", c);

fprintf(log,"Создание упорядоченного дерева:\n");

fprintf(log, " Дерево из %d элементов создано \n\n", c);

print\_file(log, d);}

else {printf("Не удалось создать дерево\n\n");

fprintf(log, " Создание упорядоченного дерева:\n");

fprintf(log, " Не удалось создать дерево\n\n");

} }

else {printf("Файл не найден\n");

fprintf(log, " Файл не найден \n");}

fprintf(log,"========================================================================\n");

GetRoot(d);

break; }

//----------------

case 'V': fprintf(log, "Вывод дерева в ширину:\n");

if (!IsEmpty(d, d.root)) vivod(d, 0, log);

else {printf("\nПустое дерево\n");

fprintf(log, "\nПустое дерево\n");

}

GetRoot(d);

//print\_file(log, d);

fprintf(log,"========================================================================\n\n");

GetRoot(d);

break;

//-----------------------

case 'D': {fprintf(log," Решение:\n");

int d1, d2,m1,m2,g1,g2;

printf("Введите указанный временной период в формате ДД.ММ.ГГ ДД.ММ.ГГ \n\n");

scanf("%2d.%2d.%4d %2d.%2d.%4d", &d1, &m1, &g1, &d2, &m2, &g2);

int m=Decide( d, d1, d2, m1, m2, g1,g2);

printf("Количество элементов, удовлетворяющих запросу: %d\n", m);

fprintf(log, " Количество элементов, удовлетворяющих запросу: %d\n\n", m);

print\_file(log, d);

fprintf(log,"========================================================================\n");

fflush(stdin);

GetRoot(d);

break;}

//---------------

case 'F':fprintf(log," Освобождение памяти:\n");

free\_tree(d);

printf("\Вся память освобождена\n\n");

fprintf(log, "\ Вся память освобождена\n\n");

print\_file(log, d);

fprintf(log,"========================================================================\n\n");

break;

case 'G': fprintf(log," Создание генеалогического дерева:\n");

f=fopen(argv[1],"r");

if (f){create\_tree(d, f, kol);

GetRoot(d);

fclose(f);

if (kol) {printf("Генеалогическое дерево успешно создано\n");

fprintf(log, " Генеалогическое дерево успешно создано\n"); }

else {printf("Не удалось создать дерево\n");

fprintf(log," Не удалось создать дерево\n");

}}

else {printf("Файл не найден\n");

fprintf(log," Файл не найден\n");}

print\_file(log, d); fprintf(log,"========================================================================\n\n");

GetRoot(d);

break;

case 'W': fprintf(log, "\n Просмотр дерева в глубину:\n");

if (!IsEmpty(d, d.root)) make\_tree(d, f, 0, log); else {printf("\nПустое дерево\n");

fprintf(log, "\nПустое дерево\n");}

GetRoot(d);

fprintf(log,"========================================================================\n\n");

GetRoot(d);

break;

case 'A': fprintf(log, "\nДобавление элемента\n");

TInfo info;

printf("Фамилия\n");

scanf("%s", info.F);

printf("Имя\n");

scanf("%s", info.I);

printf("Отчество\n");

scanf("%s", info.O);

printf("Дата\n");

scanf("%2d.%2d.%4d", &info.date.de, &info.date.m, &info.date.g);

fflush(stdin);

sort(d, info);

printf("Элемент успешно добавлен \n");

GetRoot(d);

fprintf(log, " Элемент '%s %s %s, род. %2d.%2d.%4d' успешно добавлен\n\n", info.F, info.I, info.O, info.date.de,info.date.m,info.date.g);

print\_file(log, d); fprintf(log,"========================================================================\n\n");

GetRoot(d);

break;

//-----------------------------------------------

case 'E': return 0;

break;

default:

printf("Нет такой команды\nPress any key");

getch();

}

} while (ch!='E');

return 0;

free\_tree(d);

getch();

fclose(log);

return 0;

}

1. Пример получившегося лог-файла:

Создание упорядоченного дерева:

Дерево из 10 элементов создано

Чернов Константин Михайлович, род. 11. 1.1999

Чернов Виталий Николаевич, род. 10.12.1953

\*\*

Арзамасова Елена Святославовна, род. 9. 8.1980

\*\*

Родина Елена Семеновна, род. 24.10.1961

\*\*

\*\*

Чернов Михаил Витальевич, род. 9. 3.1970

Горчакова Мария Васильевна, род. 18. 8.1924

Арзамасов Максим Устинович, род. 11. 5.1930

\*\*

Анисимова Марина Петровна, род. 19.12.1932

\*\*

\*\*

\*\*

Чернов Николай Александрович, род. 8. 4.1921

\*\*

Арзамасов Святослав Максимович, род. 1. 6.1959

\*\*

\*\*

========================================================================

Вывод дерева в ширину:

-------- Чернов Виталий Николаевич, род. 10.12.1953

-------- Арзамасова Елена Святославовна, род. 9. 8.1980

-------- Родина Елена Семеновна, род. 24.10.1961

Чернов Константин Михайлович, род. 11. 1.1999

-------- Арзамасов Максим Устинович, род. 11. 5.1930

-------- Анисимова Марина Петровна, род. 19.12.1932

-------- Горчакова Мария Васильевна, род. 18. 8.1924

-------- Чернов Михаил Витальевич, род. 9. 3.1970

-------- Чернов Николай Александрович, род. 8. 4.1921

-------- Арзамасов Святослав Максимович, род. 1. 6.1959

========================================================================

Добавление элемента:

Элемент 'Иванов Петр Иванович, род. 11.12.1965' успешно добавлен

Чернов Константин Михайлович, род. 11. 1.1999

Чернов Виталий Николаевич, род. 10.12.1953

\*\*

Арзамасова Елена Святославовна, род. 9. 8.1980

\*\*

Родина Елена Семеновна, род. 24.10.1961

\*\*

\*\*

Чернов Михаил Витальевич, род. 9. 3.1970

Горчакова Мария Васильевна, род. 18. 8.1924

Арзамасов Максим Устинович, род. 11. 5.1930

\*\*

Анисимова Марина Петровна, род. 19.12.1932

\*\*

\*\*

\*\*

Чернов Николай Александрович, род. 8. 4.1921

\*\*

Арзамасов Святослав Максимович, род. 1. 6.1959

Иванов Петр Иванович, род. 11.12.1965

\*\*

\*\*

\*\*

========================================================================

Добавление элемента:

Элемент 'Арсеньев Максим Максимович, род. 16. 9.1934' успешно добавлен

Чернов Константин Михайлович, род. 11. 1.1999

Чернов Виталий Николаевич, род. 10.12.1953

\*\*

Арзамасова Елена Святославовна, род. 9. 8.1980

\*\*

Родина Елена Семеновна, род. 24.10.1961

\*\*

\*\*

Чернов Михаил Витальевич, род. 9. 3.1970

Горчакова Мария Васильевна, род. 18. 8.1924

Арзамасов Максим Устинович, род. 11. 5.1930

\*\*

Анисимова Марина Петровна, род. 19.12.1932

Арсеньев Максим Максимович, род. 16. 9.1934

\*\*

\*\*

\*\*

\*\*

Чернов Николай Александрович, род. 8. 4.1921

\*\*

Арзамасов Святослав Максимович, род. 1. 6.1959

Иванов Петр Иванович, род. 11.12.1965

\*\*

\*\*

\*\*

========================================================================

Вывод дерева в ширину:

-------- Чернов Виталий Николаевич, род. 10.12.1953

-------- Арзамасова Елена Святославовна, род. 9. 8.1980

-------- Родина Елена Семеновна, род. 24.10.1961

Чернов Константин Михайлович, род. 11. 1.1999

-------- Арзамасов Максим Устинович, род. 11. 5.1930

-------- Арсеньев Максим Максимович, род. 16. 9.1934

-------- Анисимова Марина Петровна, род. 19.12.1932

-------- Горчакова Мария Васильевна, род. 18. 8.1924

-------- Чернов Михаил Витальевич, род. 9. 3.1970

-------- Чернов Николай Александрович, род. 8. 4.1921

-------- Иванов Петр Иванович, род. 11.12.1965

-------- Арзамасов Святослав Максимович, род. 1. 6.1959

========================================================================

Освобождение памяти:

Вся память освобождена

\*\*

========================================================================

Создание генеалогического дерева:

Генеалогическое дерево успешно создано

Чернов Константин Михайлович, род. 11. 1.1999

Чернов Михаил Витальевич, род. 9. 3.1970

Чернов Виталий Николаевич, род. 10.12.1953

Чернов Николай Александрович, род. 8. 4.1921

\*\*

\*\*

Горчакова Мария Васильевна, род. 18. 8.1924

\*\*

\*\*

\*\*

Арзамасова Елена Святославовна, род. 9. 8.1980

Арзамасов Святослав Максимович, род. 1. 6.1959

Арзамасов Максим Устинович, род. 11. 5.1930

\*\*

\*\*

Анисимова Марина Петровна, род. 19.12.1932

\*\*

\*\*

Родина Елена Семеновна, род. 24.10.1961

\*\*

\*\*

========================================================================

Просмотр дерева в глубину:

Чернов Константин Михайлович, род. 11. 1.1999

-------- Чернов Михаил Витальевич, род. 9. 3.1970

-------- Чернов Виталий Николаевич, род. 10.12.1953

-------- Чернов Николай Александрович, род. 8. 4.1921

--------

--------

-------- Горчакова Мария Васильевна, род. 18. 8.1924

--------

--------

--------

-------- Арзамасова Елена Святославовна, род. 9. 8.1980

-------- Арзамасов Святослав Максимович, род. 1. 6.1959

-------- Арзамасов Максим Устинович, род. 11. 5.1930

--------

--------

-------- Анисимова Марина Петровна, род. 19.12.1932

--------

--------

-------- Родина Елена Семеновна, род. 24.10.1961

--------

--------

========================================================================

Решение:

Количество элементов, удовлетворяющих запросу: 4

Чернов Константин Михайлович, род. 11. 1.1999

Чернов Михаил Витальевич, род. 9. 3.1970

Чернов Виталий Николаевич, род. 10.12.1953

Чернов Николай Александрович, род. 8. 4.1921

\*\*

\*\*

Горчакова Мария Васильевна, род. 18. 8.1924

\*\*

\*\*

\*\*

Арзамасова Елена Святославовна, род. 9. 8.1980

Арзамасов Святослав Максимович, род. 1. 6.1959

Арзамасов Максим Устинович, род. 11. 5.1930

\*\*

\*\*

Анисимова Марина Петровна, род. 19.12.1932

\*\*

\*\*

Родина Елена Семеновна, род. 24.10.1961

\*\*

\*\*

========================================================================

Пример входного файла

Чернов

Константин

Михайлович

11

01

1999

Чернов

Михаил

Витальевич

09

03

1970

Чернов

Виталий

Николаевич

10

12

1953

Чернов

Николай

Александрович

08

04

1921

\*\*

\*\*

Горчакова

Мария

Васильевна

18

08

1924

\*\*

\*\*

\*\*

Арзамасова

Елена

Святославовна

09

08

1980

Арзамасов

Святослав

Максимович

01

06

1959

Арзамасов

Максим

Устинович

11

05

1930

\*\*

\*\*

Анисимова

Марина

Петровна

19

12

1932

\*\*

\*\*

Родина

Елена

Семеновна

24

10

1961

\*\*

\*\*