Министерство образования Республики Беларусь

Учреждение образования

«Гомельский государственный университет

имени Франциска Скорины»

Математический факультет

Кафедра вычислительной математики и программирования

**Структуры и алгоритмы обработки данных**

**Контрольная работа**

**Вариант 1**

Исполнитель:

студент группы ПОCЗ-31 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Барабанов А.П.

Научный руководитель:

доц. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Демуськов А.Б.

**Гомель 2022**

## Индивидуальное задание

|  |  |  |
| --- | --- | --- |
| 1 | Отдел кадров | фамилия сотрудника, имя, отчество, должность, стаж работы, оклад |

## Описание методов решения

### Представление в памяти

Структура элемента данных содержит шесть полей: 4 строковых и два числовых:

/\*\*

\* Сотрудник.

\*/

struct person\_rec

{

char last[MAX\_NAME\_SIZE]; /\* фамилия \*/

char first[MAX\_NAME\_SIZE]; /\* имя \*/

char patronymic[MAX\_NAME\_SIZE]; /\* отчество \*/

char position[MAX\_POS\_SIZE]; /\* должность \*/

unsigned tenure; /\* стаж \*/

unsigned salary; /\* оклад \*/

};

typedef struct person\_rec Employee;

"База данных" в оперативной памяти представляется в виде двунаправленного линейного списка. Элемент списка задается структурой с данными и указателями на предыдущий и следующий элементы.

typedef struct item\_rec Item;

struct item\_rec

{

Employee data; /\* Данные \*/

Item\* prev; /\* Предыдущий элемент \*/

Item\* next; /\* Следующий элемент \*/

};

Список содержит указатели на первый и последний элементы, а также размер для его быстрого определения.

typedef struct list\_rec

{

unsigned size;

Item\* first;

Item\* last;

} List;

### Модульная структура программного изделия

Программное изделие выполняется в виде четырех модулей.

Модуль I/O содержит вспомогательные функции для ввода/вывода, которые применяются в программе. Они описаны в заголовочном файле IO.h, их реализация – в файле IO.c.

Модуль Data описывает тип данных, который используется в программе, а также функции сравнения записей по различным полям. Описания содержатся в файле Data.h, реализация – в файле Data.c.

Модуль List описывает связный список и дополнительную информацию, такую как имя загруженного файла и флаг изменения. Описания содержатся в файле List.h, реализация – в файле List.c.

Наконец, главный модуль Laba содержит функционал для организации диалога с пользователем и исполнения введенных им команд. Его исходный текст находится в файле Laba.c.

## Описание логической структуры

### Главный модуль

Функции управления программой

static int init( void );

Инициализирует "базу данных". Возвращает 0 в случае успеха.

static unsigned readCommand( void );

Выводит меню и возвращает введенную пользователем команду

static int executeCommand( unsigned command );

Обрабатывает команду пользователя, вызывая одну из нижеописанных функций команд.

static void done( void );

Освобождает выделенную по ходу работы память.

Функции команд:

static int quit( void );

Функция проверяет, изменялась ли "база данных", и если да, то запрашивает у пользователя необходимость ее сохранения в файл. Возвращает TRUE, если пришло время завершать работу.

static int load( void );

Запрашивает у пользователя имя файла и пытается его считать в новый список. В случае успеха старый заменяется на новый, в случае ошибки чтения никаких изменений в оригинальном списке не производится .

static int save( void );

Заправшивает у пользователя имя файла и записывает список в файл.

static int view( void );

Выводит на экран все записи в списке.

static int sort( void );

Запрашивает у пользователя поле для сортировки и нужно ли сортировать по убыванию. Сортирует список в соответствии с выбранным критерием.

static int add( void );

Запрашивает у пользователя поля для новой записи и добавляет ее в список в конец.

static int del( void );

Запрашивает у пользователя индекс записи (его можно узнать с помощью команды просмотра) и удаляет ее из списка.

static int edit( void );

Запрашивает у пользователя индекс записи, и последовательно новые значения для полей, отображая старые, которые можно оставить без изменения вводом пустой строки.

### Модуль данных

int compareNames( const Employee\* e1, const Employee\* e2 );

int compareTenures( const Employee\* e1, const Employee\* e2 );

int compareSalaries( const Employee\* e1, const Employee\* e2 );

Функции сравнивают соответствующие поля и возвращают значение <0, если первый аргумент должен идти раньше второго в порядке возрастания, >0 если позже, и 0 если значения полей равны. Используется для сортировки.

### Модуль ввода/вывода

int readLine( FILE\* file, char\* buffer, unsigned buffer\_size );

Читает строку в буфер с заданным размером, гарантируя завершение строки 0 и невыход за пределы буфера.

int askYNC( const char\* question );

Выводит запрос и ожидает ввода Y/y для выбора утвердительного ответа, N/n для выбора отрицательного ответа и любого другого символа для отмены действия.

void waitEnter( void );

Выводит приглашение и ожидает нажатия клавиши <ENTER>.

int readUnsigned( const char\* what, unsigned\* value );

int readStr( const char\* what, char\* value, unsigned value\_size );

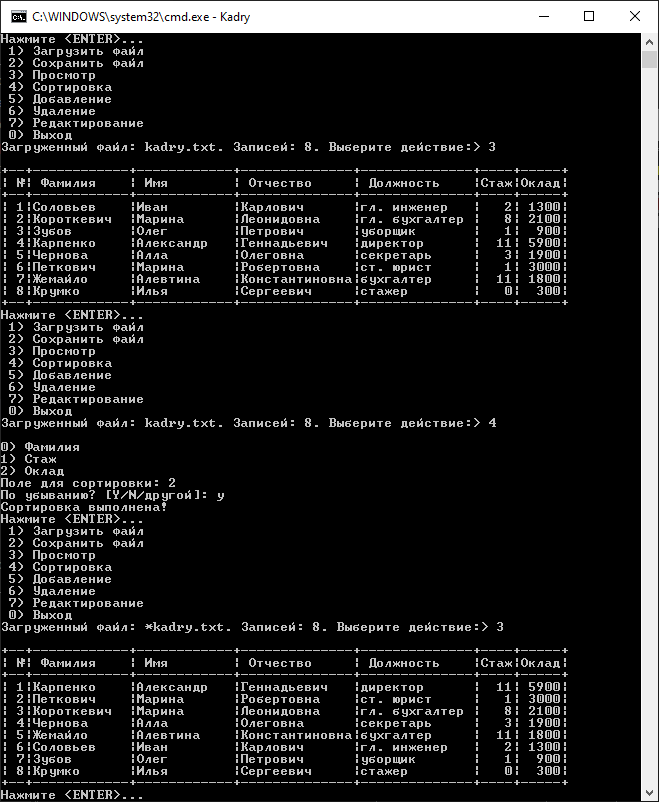
Ввод значений соответствующего типа в переменную value. Возвращается 0 в случае успеха и не 0 в случае ошибки.

int readUnsignedV( const char\* what, unsigned\* value, unsigned oldvalue );

int readStrV( const char\* what, char\* value, unsigned value\_size, const char\* oldvalue );

Ввод значений соответствующего типа с возможностью оставить старое в случае ввода пустой строки.

## Пример работы программы



**Код программы:**

#include <stdio.h>

#include <locale.h>

#include "IO.h"

#include "Data.h"

#include "List.h"

/\*----------------------------------------------------------------------------

КОНСТАНТЫ

----------------------------------------------------------------------------\*/

/\*\*

\* Пункты меню.

\*/

enum {

QUIT,

LOAD,

SAVE,

VIEW,

SORT,

ADD,

REMOVE,

EDIT,

MENU\_SIZE

};

/\*----------------------------------------------------------------------------

ТИПЫ

----------------------------------------------------------------------------\*/

/\*\*

\* Тип функции, выполняющей команду меню.

\* Возвращает 0 если нужен выход иначе не 0.

\*/

typedef int (\*CommandFunc)( void );

/\*----------------------------------------------------------------------------

ФУНКЦИИ ВЫПОЛНЕНИЯ КОМАНД

----------------------------------------------------------------------------\*/

static int quit( void ); /\* выход из программы \*/

static int load( void ); /\* загрузка файла \*/

static int save( void ); /\* сохранение файла \*/

static int view( void ); /\* просмотр записей \*/

static int sort( void ); /\* сортировка \*/

static int add( void ); /\* добавление записи \*/

static int del( void ); /\* удаление записи \*/

static int edit( void ); /\* редактирование записи \*/

/\*----------------------------------------------------------------------------

ФУНКЦИИ УПРАВЛЕНИЯ

----------------------------------------------------------------------------\*/

static int init( void ); /\* инициализация \*/

static unsigned readCommand( void ); /\* чтение команды \*/

static int executeCommand( unsigned command ); /\* ее выполнение \*/

static void done( void ); /\* очистка \*/

/\*----------------------------------------------------------------------------

ПЕРЕМЕННЫЕ

----------------------------------------------------------------------------\*/

/\*\*

\* Массив меток команд.

\*/

static const char\* COMMAND\_LABELS[MENU\_SIZE] = {

"Выход",

"Загрузить файл",

"Сохранить файл",

"Просмотр",

"Сортировка",

"Добавление",

"Удаление",

"Редактирование"

};

/\*\*

\* Массив функций команд.

\*/

static CommandFunc COMMAND\_PROCESSORS[MENU\_SIZE] = {

quit,

load,

save,

view,

sort,

add,

del,

edit

};

/\*\*

\* Названия полей.

\*/

static const char\* FIELD\_NAMES[NUM\_FIELDS] = {

"Фамилия",

"Стаж",

"Оклад"

};

/\*----------------------------------------------------------------------------

ГЛАВНАЯ ФУНКЦИЯ

----------------------------------------------------------------------------\*/

int main( int argc, char\*\* argv )

{

int retval; /\* Код возврата, 0 = ОК \*/

setlocale(LC\_ALL,"RUS"); /\* Устанавливаем кодовую страницу, хотя

можно просто писать исходники в cp866 \*/

if( !(retval = init()) ) /\* Если инициализация прошла успешно \*/

/\* Пока команда не вернула код выхода, выполняем команды \*/

while( executeCommand(readCommand()) );

done(); /\* Очистка \*/

return retval;

}

/\*----------------------------------------------------------------------------

ЛОКАЛЬНЫЕ ФУНКЦИИ

----------------------------------------------------------------------------\*/

/\*\*

\* Печать пункта меню с индексом index.

\*/

static void printCmd( unsigned index )

{

printf("%2d) %s\n",index,COMMAND\_LABELS[index]);

}

/\*\*

\* Печать приглашения для ввода команды.

\*/

static void printPrompt( void )

{

const char\* filename = getFileName();

int changed = isListChanged();

printf("Загруженный файл: %s%s. Записей: %u. Выберите действие:> ",

changed?"\*":"",

filename?filename:"(-)",

getListSize());

}

/\*\*

\* Печать линии для рамки.

\*/

static void printl( void )

{

printf("+--+------------"

"+------------+--------------"

"+--------------+----+-----+\n");

}

/\*\*

\* Печать заголовка таблицы.

\*/

static void printh( void )

{

printf("| №| Фамилия "

"| Имя "

"| Отчество "

"| Должность |Стаж|Оклад|\n");

}

/\*\*

\* Печать записи pdata с индексом index.

\*/

static void print( unsigned index, const Employee\* pdata )

{

printf("|%2d|%-12s|%-12s|%-14s|%-14s|%4u|%5u|\n",

index,

pdata->last,

pdata->first,

pdata->patronymic,

pdata->position,

pdata->tenure,

pdata->salary);

}

/\*\*

\* Запрос у пользователя индекса поля для сортировки таблицы.

\* В случае ошибки возвращает нелегальный индекс.

\*/

static unsigned getField( void )

{

unsigned i, retval;

for( i=0; i<NUM\_FIELDS; ++i ) {

printf("%d) %s\n",i,FIELD\_NAMES[i]);

}

if( readUnsigned("Поле для сортировки",&retval) )

return NUM\_FIELDS;

else

return retval;

}

/\*----------------------------------------------------------------------------

РЕАЛИЗАЦИЯ ФУНКЦИЙ УПРАВЛЕНИЯ

----------------------------------------------------------------------------\*/

/\*\*

\* Инициализация объектов.

\*/

static int init( void )

{

return openList();

}

/\*\*

\* Вывод меню и ввод команды.

\*/

static unsigned readCommand( void )

{

unsigned ret, cmd;

for( cmd=1; cmd<MENU\_SIZE; ++cmd )

printCmd(cmd);

printCmd(QUIT);

printPrompt();

scanf("%u",&ret);

printf("\n");

fflush(stdin);

return ret;

}

/\*\*

\* Возвращает 0, если надо остановить работу.

\*/

static int executeCommand( unsigned command )

{

int ret;

if( command < MENU\_SIZE ) {

ret = COMMAND\_PROCESSORS[command]();

} else {

puts("\* Неверная команда!");

ret = 1;

}

if( ret ) waitEnter(); /\* Ожидание нажатия ENTER \*/

return ret;

}

/\*\*

\* Освобождение памяти и пр.

\*/

static void done( void )

{

closeList();

puts("Bye!");

}

/\*----------------------------------------------------------------------------

РЕАЛИЗАЦИЯ ФУНКЦИЙ КОМАНД

----------------------------------------------------------------------------\*/

/\*\*

\* Команда выхода.

\*/

static int quit( void )

{

int answer;

/\*

Если список был изменен, запрос на сохранение изменений.

Если подтвержден, сохраняем, если отменен, ничего не делаем,

иначе выходим без сохранения.

\*/

if( isListChanged() ) {

answer = askYNC("Сохранить изменения перед выходом");

if( answer < 0 ) {

puts("Выход отменен");

return 1; /\* продолжаем работу \*/

} else if( answer > 0 ) {

puts("Сохраняем...");

return save(); /\* если ошибка сохранения, продолжаем \*/

}

}

return 0; /\* сигнал, что пора на выход \*/

}

/\*\*

\* Команда восстановления из файла.

\*/

static int load( void )

{

char filename[MAX\_PATH\_SIZE];

if( readStr("Введите имя файла",filename,MAX\_PATH\_SIZE) )

puts("\* Ошибка ввода имени!");

else {

if( loadList(filename) )

printf("\* Ошибка загрузки файла %s!\n",filename);

else {

printf("Файл %s загружен!\n",filename);

}

}

return 1;

}

/\*\*

\* Команда сохранения файла.

\*/

static int save( void )

{

char filename[MAX\_PATH\_SIZE];

if( readStr("Введите имя файла",filename,MAX\_PATH\_SIZE) )

puts("\* Ошибка ввода имени!");

else {

if( saveList(filename) )

printf("\* Ошибка записи файла %s!\n",filename);

else {

printf("Файл %s записан!\n",filename);

}

}

return 1;

}

/\*\*

\* Команда просмотра таблицы с элементами списка.

\*/

static int view( void )

{

printl(); /\* линия \*/

printh(); /\* заголовок \*/

printl(); /\* линия \*/

forEach(print); /\* записи \*/

printl(); /\* линия \*/

return 1; /\* продолжаем работу \*/

}

/\*\*

\* Команда сортировки.

\* Запрашивает поле и если оно легально, направление

\* сортировки, и сортирует массив.

\*/

static int sort( void )

{

unsigned field; /\* индекс поля для сортировки \*/

int desc; /\* флаг сортировки по убыванию \*/

field = getField();

desc = askYNC("По убыванию");

if( desc < 0 ) {

puts("Сортировка отменена!");

} else {

if( sortList(field,desc) ) {

puts("\* Ошибка сортировки!");

} else {

puts("Сортировка выполнена!");

}

}

return 1; /\* продолжаем работу \*/

}

/\*\*

\* Функция добавления записи.

\*/

static int add( void )

{

Employee newrec; /\* Новая запись \*/

int error = 0; /\* Код ошибки != 0 \*/

puts("Вводите новые значения:");

if( !error )

error = readStr("Фамилия",newrec.last,MAX\_NAME\_SIZE);

if( !error )

error = readStr("Имя",newrec.first,MAX\_NAME\_SIZE);

if( !error )

error = readStr("Отчество",newrec.patronymic,MAX\_NAME\_SIZE);

if( !error )

error = readStr("Должность",newrec.position,MAX\_POS\_SIZE);

if( !error )

error = readUnsigned("Стаж",&newrec.tenure);

if( !error )

error = readUnsigned("Оклад",&newrec.salary);

if( !error )

puts("Запись добавлена!");

else

puts("\* Ошибка добавления!");

return 1; /\* продолжаем работу \*/

}

/\*\*

\* Команда удаления записи.

\* Запрашивается индекс (отображается в списке).

\*/

static int del( void )

{

unsigned index;

if( readUnsigned("Введите индекс элемента",&index) )

puts("\* Ошибка ввода индекса!");

else {

if( removeItem(index) )

puts("\* Ошибка удаления записи!");

else

puts("Запись удалена!");

}

return 1; /\* продолжаем работу \*/

}

/\*\*

\* Команда редактирования записи.

\* Запрашивается индекс элемента.

\* Запрашиваются все поля по очереди, при вводе пустой строки

\* значение не меняется.

\*/

static int edit( void )

{

Employee newrec; /\* Новая запись \*/

unsigned index; /\* Индекс редактируемой записи \*/

int error = 0; /\* Код ошибки != 0 \*/

if( readUnsigned("Введите индекс элемента",&index) )

puts("\* Ошибка ввода индекса!");

else {

Employee\* prec = getItem(index);

if( prec ) { /\* Если индекс нелегальный, будет NULL \*/

puts("Вводите новые значения (пустая строка - оставить старые)");

/\* Далее читаем по очереди все поля при отсутствии ошибок ввода \*/

if( !error )

error = readStrV("Фамилия",newrec.last,MAX\_NAME\_SIZE,prec->last);

if( !error )

error = readStrV("Имя",newrec.first,MAX\_NAME\_SIZE,prec->first);

if( !error )

error = readStrV("Отчество",newrec.patronymic,MAX\_NAME\_SIZE,prec->patronymic);

if( !error )

error = readStrV("Должность",newrec.position,MAX\_POS\_SIZE,prec->position);

if( !error )

error = readUnsignedV("Стаж",&newrec.tenure,prec->tenure);

if( !error )

error = readUnsignedV("Оклад",&newrec.salary,prec->salary);

if( !error ) {

\*prec = newrec; /\* Заменяем запись на новую \*/

puts("Запись изменена!");

} else /\* При ошибке выводим сообщение \*/

puts("\* Ошибка ввода значений!");

} else

printf("\* Ошибка доступа к записи №%u!\n",index);

}

return 1; /\* продолжаем работу \*/

}

Команды для работы с набором данных:

#include <string.h>

#include <stdio.h>

#include <stdlib.h>

#include "IO.h"

#include "Data.h"

#include "List.h"

/\*\*

\* Элемент списка.

\*/

typedef struct item\_rec Item;

struct item\_rec

{

Employee data; /\* Данные \*/

Item\* prev; /\* Предыдущий элемент \*/

Item\* next; /\* Следующий элемент \*/

};

/\*\*

\* Список.

\*/

typedef struct list\_rec

{

unsigned size; /\* Число элементов \*/

Item\* first; /\* Указатель на первый элемент \*/

Item\* last; /\* Указатель на последний элемент \*/

} List;

static char current\_filename[MAX\_PATH\_SIZE]; /\* Текущий загруженный файл \*/

static int loaded\_flag = 0; /\* Флаг не 0, если файл загружен \*/

static int changed\_flag = 0; /\* Флаг не 0, если массив изменен \*/

static List list; /\* Список \*/

/\*\*

\* Копирование строки src в буфер buffer размером buffer\_size

\* (типа strncpy. но гарантируется легальная строка с 0 в конце).

\*/

static void copyMax( char\* buffer, unsigned buffer\_size, const char\* src )

{

unsigned len = strlen(src);

unsigned buffer\_max = buffer\_size - 1;

unsigned min = buffer\_max < len ? buffer\_max : len;

unsigned i;

for( i=0; i<min; ++i )

buffer[i] = src[i];

buffer[min] = '\0';

}

/\*\*

\* Разбор строки str и занесение значений, разделенных |,

\* в запись data.

\* Возвращает 0 при успехе, 1 при ошибке.

\*/

static int parseRec( char\* str, Employee\* data )

{

char\* oldptr = str;

char\* ptr = strchr(oldptr,'|'); if( !ptr ) return 1; \*ptr = '\0';

copyMax(data->last,MAX\_NAME\_SIZE,oldptr);

oldptr = ++ptr;

ptr = strchr(oldptr,'|'); if( !ptr ) return 1; \*ptr = '\0';

copyMax(data->first,MAX\_NAME\_SIZE,oldptr);

oldptr = ++ptr;

ptr = strchr(oldptr,'|'); if( !ptr ) return 1; \*ptr = '\0';

copyMax(data->patronymic,MAX\_NAME\_SIZE,oldptr);

oldptr = ++ptr;

ptr = strchr(oldptr,'|'); if( !ptr ) return 1; \*ptr = '\0';

copyMax(data->position,MAX\_POS\_SIZE,oldptr);

oldptr = ++ptr;

ptr = strchr(oldptr,'|'); if( !ptr ) return 1; \*ptr = '\0';

if( sscanf(oldptr,"%u",&data->tenure) != 1 ) return 1;

oldptr = ++ptr;

if( sscanf(oldptr,"%u",&data->salary) != 1 ) return 1;

return 0;

}

/\*\*

\* Добавление записи prec к списку plist.

\*/

static void addToList( List\* plist, const Employee\* prec )

{

Item\* newrec = (Item\*)malloc(sizeof(Item)); /\* Новый элемент \*/

newrec->data = \*prec;

newrec->prev = plist->last; /\* Предыдущий - это бывший последний \*/

if( plist->last ) plist->last->next = newrec; /\* а бывший последний элемент стал предпоследним \*/

plist->last = newrec; /\* добавленный элемент будет последним \*/

newrec->next = 0; /\* следующего пока нет \*/

if( !plist->first ) plist->first = newrec; /\* если первый элемент, ставим указатель \*/

++plist->size; /\* увеличиваем размер \*/

}

/\*\*

\* Удаление записи с индексом index из списка plist.

\*/

static void removeFromList( List\* plist, unsigned index )

{

Item\* cur = plist->first;

unsigned i = 0;

while( ++i < index ) cur = cur->next; /\* переход к нужному индексу \*/

if( cur->prev ) cur->prev->next = cur->next; /\* сменяем указатель в предыдущем \*/

if( cur->next ) cur->next->prev = cur->prev; /\* сменяем указатель в следующем \*/

if( plist->first == cur ) plist->first = cur->next; /\* сменяем если надо указатель на первый \*/

if( plist->last == cur ) plist->last = cur->prev; /\* сменяем если надо указатель на последний \*/

free(cur); /\* Освобождаем память записи удаленного элемента \*/

--plist->size; /\* уменьшаем размер \*/

}

/\*\*

\* Начальная инициализация списка, вызвать перед началом работы с ним.

\*/

static void initList( List\* plist )

{

plist->size = 0;

plist->first = 0;

plist->last = 0;

}

/\*\*

\* Очистка памяти списка, вызвать в конце работы.

\*/

static void clearList( List\* plist )

{

Item\* cur = plist->first;

Item\* next;

while( cur != 0 ) {

next = cur->next;

free(cur);

cur = next;

}

initList(plist);

}

/\*\*

\* Функция доступа к мени текущего файла, если не был загружен, 0.

\*/

const char\* getFileName( void )

{

return loaded\_flag ? current\_filename : 0;

}

/\*\*

\* Был ли изменен список (добавлено, удалено, отредактировано...)?

\*/

int isListChanged( void )

{

return changed\_flag;

}

/\*\*

\* Загрузка списка из файла fname.

\*/

int loadList( const char\* fname )

{

FILE\* f; /\* Файл \*/

char buffer[256]; /\* Буфер для строк \*/

int fileok; /\* Флаг отсутствия ошибок \*/

Employee newrec; /\* Очередная запись \*/

List newlist; /\* Новый список \*/

initList(&newlist);

f = fopen(fname,"rt");

if( f != NULL ) {

fileok = 1;

while( fileok && (!feof(f)) ) { /\* пока нет ошибок и не конец файла \*/

if( readLine(f,buffer,256) ) { /\* читаем строку \*/

fileok = 0; /\* если ошибка, сбросим флаг отсутствия ошибок \*/

} else {

if( strlen(buffer)>0 ) { /\* если строка не пустая, \*/

if( parseRec(buffer,&newrec) ) /\* обработаем запись \*/

fileok = 0;

else

addToList(&newlist,&newrec); /\* и добавим ее в список \*/

}

}

}

fclose(f); /\* закрытие файла \*/

if( fileok ) {

/\* Если прочитали ОК заменяем старый список на новый \*/

clearList(&list);

list = newlist;

loaded\_flag = 1;

changed\_flag = 0;

strcpy(current\_filename,fname);

} else {

/\* Иначе удаляем новый список, старый остается \*/

clearList(&newlist);

}

return !fileok;

} else

return 1;

}

/\*\*

\* Сохранение списка в файл.

\* Пишем по очереди все поля всех записей.

\* Запись = 1 строка.

\* Поля разделены |.

\*/

int saveList( const char\* fname )

{

FILE\* f;

Item\* cur;

f = fopen(fname,"wt");

if( f ) {

cur = list.first;

while( cur ) {

fprintf(f,"%s|%s|%s|%s|%u|%u\n",

cur->data.last,

cur->data.first,

cur->data.patronymic,

cur->data.position,

cur->data.tenure,

cur->data.salary);

cur = cur->next;

}

fclose(f);

strcpy(current\_filename,fname);

changed\_flag = 0;

return 0;

} else

return 1;

}

/\*\*

\* Начало работы. Инициализация списка.

\*/

int openList( void )

{

initList(&list);

return 0; /\* Всегда успешно \*/

}

/\*\*

\* Конец работы.

\*/

void closeList( void )

{

clearList(&list);

}

/\*\*

\* Размер списка.

\*/

unsigned getListSize( void )

{

return list.size;

}

/\*\*

\* Доступ к записям по индексу.

\*/

Employee\* getItem( unsigned index )

{

Item\* cur;

unsigned i;

if( (index==0) || (index>list.size) )

return 0;

cur = list.first;

while( ++i < index ) cur = cur->next;

return &cur->data;

}

/\*\*

\* Добавление новой записи.

\*/

int addItem( const Employee\* pitem )

{

addToList(&list,pitem);

changed\_flag = 1;

return 0;

}

/\*\*

\* Удаление записи по индексу 1..размер.

\* При нелегальном индексе ошибка.

\*/

int removeItem( unsigned index )

{

if( (index==0) || (index>list.size) )

return 1;

else {

removeFromList(&list,index);

changed\_flag = 1;

return 0;

}

}

/\*\*

\* Вспомогательная функция обмена элементов списка i1 и i2.

\* Используется в сортировках.

\*/

static void swap( Item\* i1, Item\* i2 )

{

Employee tmp = i1->data;

i1->data = i2->data;

i2->data = tmp;

}

/\*\*

\* Сортировка списка по полю с индексом field

\* при desc != 0 по убыванию, иначе возрастанию.

\*/

int sortList( unsigned field, int descending )

{

CompareFunc compare;

Item \*i, \*j;

int comp\_result;

if( list.size > 1 ) { /\* Если есть что сортировать \*/

switch( field ) {

case F\_NAME: compare = compareNames; break;

case F\_TENURE: compare = compareTenures; break;

case F\_SALARY: compare = compareSalaries; break;

default: return 1; /\* Нелегальное поле! \*/

}

i = list.first;

while( i->next ) { /\* Сортировка простым выбором \*/

j = i->next;

while( j ) {

/\* Сравниваем, если по убыванию, обращаем результат \*/

comp\_result = compare(&i->data,&j->data);

if( descending ) comp\_result = -comp\_result;

/\* Если в неправильном порядке, обмениваем \*/

if( comp\_result > 0 ) {

swap(i,j);

}

j = j->next;

}

i = i->next;

}

}

changed\_flag = 1; /\* Сортировка изменяет массив \*/

return 0;

}

/\*\*

\* Выполняет функцию func на каждой из записей в списке.

\* Функция получает указатель на запись и ее индекс (1..размер).

\*/

void forEach( VisitorFunc func )

{

Item\* cur;

unsigned index = 0;

if( list.first ) {

cur = list.first;

while( cur ) {

func(++index,&cur->data);

cur = cur->next;

}

}

}

Функции ввода-вывода:

#include <stdio.h>

#include <string.h>

#include "IO.h"

/\*\*

\* Конвертация ДОС-кодировки, которая используется при вводе независимо

\* от наcтроек locale, в Windows-кодировку, которая используется при выводе

\* на консоль при установке локали в "RUS".

\*/

static void fixInput( char\* str )

{

unsigned i = 0;

while( str[i]!=0 ) {

unsigned char ch = str[i];

if( (ch >= 0x80) && (ch<0xB0) )

str[i] = ch + 0x40;

else if( (ch >= 0xE0) && (ch<0xF0) )

str[i] = ch + 0x10;

else if( ch == 0xF0 ) /\* Ё \*/

str[i] = (char)0xA8;

else if( ch == 0xF0 ) /\* ё \*/

str[i] = (char)0xB8;

++i;

}

}

/\*\*

\* Чтение строки из файла file в буфер buffer размером buffer\_size байт.

\* Возвращает !=0 при ошибке.

\*/

int readLine( FILE\* file, char\* buffer, unsigned buffer\_size )

{

unsigned offset;

if( buffer\_size == 0 ) return 1;

if( fgets(buffer,buffer\_size,file) ) {

offset = 0;

while( (buffer[offset]!='\r')

&& (buffer[offset]!='\n')

&& (buffer[offset]!='\0') )

++offset;

buffer[offset] = '\0';

return 0;

} else {

buffer[0] = '\0';

return !feof(file);

}

}

/\*\*

\* Ввод строки с консоли в буфер buffer размером buffer\_size байт.

\* Нужно очистить буфер ввода от оставшихся знаков перевода строки

\* и исправить символы русских букв, поскольку ввод работает в

\* кодировке cp866, даже если вывод установлен в windows-1251.

\*/

int inputLine( char\* buffer, unsigned buffer\_size )

{

int retval = readLine(stdin,buffer,buffer\_size);

fflush(stdin);

if( !retval )

fixInput(buffer);

return retval;

}

/\*\*

\* Запрос ответа Да/Нет/Отмена на вопрос question.

\* Да = 1

\* Нет = 0

\* Отмена = -1

\*/

int askYNC( const char\* question )

{

char buf[16];

printf("%s? [Y/N/другой]: ",question);

if( !readLine(stdin,buf,16) ) {

if( (buf[0]=='y') || (buf[0]=='Y') )

return 1;

if( (buf[0]=='n') || (buf[0]=='N') )

return 0;

}

return -1;

}

/\*\*

\* Ожидание нажатия клавиши ENTER.

\*/

void waitEnter( void )

{

char buf[4];

printf("Нажмите <ENTER>...");

fflush(stdin);

readLine(stdin,buf,4);

fflush(stdin);

}

/\*\*

\* Ввод беззнакового значения в value.

\* Выводится приглашение what.

\*/

int readUnsigned( const char\* what, unsigned\* value )

{

char buffer[128];

readStr(what,buffer,128);

return sscanf(buffer,"%u",value) != 1;

}

/\*\*

\* Ввод сроки в буфер value размером value\_size.

\* Выводится приглашение what.

\*/

int readStr( const char\* what, char\* value, unsigned value\_size )

{

printf("%s: ",what);

return inputLine(value,value\_size);

}

/\*\*

\* Ввод беззнакового значения в value.

\* Выводится приглашение what.

\* При вводе пустой строки копируется oldvalue.

\*/

int readUnsignedV( const char\* what, unsigned\* value, unsigned oldvalue )

{

char buffer[128];

int res;

printf("%s (%u): ",what,oldvalue);

res = inputLine(buffer,128);

if( res ) return res;

if( strlen(buffer)==0 ) {

\*value = oldvalue;

return 0;

} else

return sscanf(buffer,"%u",value) != 1;

}

/\*\*

\* Ввод строки в буфер value размером buffer\_size.

\* Выводится приглашение what.

\* При вводе пустой строки копируется oldvalue.

\*/

int readStrV( const char\* what, char\* value, unsigned value\_size, const char\* oldvalue )

{

int res;

unsigned len, i, min;

printf("%s (%s): ",what,oldvalue);

res = inputLine(value,value\_size);

if( res ) return res;

if( strlen(value)==0 ) {

len = strlen(oldvalue);

min = len < value\_size-1 ? len : value\_size-1;

for( i=0; i<min; ++i )

value[i] = oldvalue[i];

value[min] = '\0';

}

return res;

}

Тип данных и операции над ними:

#include <string.h>

#include "Data.h"

/\*\*

\* Функция сравнения имен (фамилий на самом деле).

\*/

int compareNames( const Employee\* e1, const Employee\* e2 )

{

return strcmp(e1->last,e2->last);

}

/\*\*

\* Функция сравнения стажей.

\*/

int compareTenures( const Employee\* e1, const Employee\* e2 )

{

return e1->tenure < e2->tenure ? -1 : e1->tenure > e2->tenure ? 1 : 0;

}

/\*\*

\* Функция сравнения зарплат.

\*/

int compareSalaries( const Employee\* e1, const Employee\* e2 )

{

return e1->salary < e2->salary ? -1 : e1->salary > e2->salary ? 1 : 0;

}