Assignment: Create an in-memory Linked List.

Do not use list C++ list objects. This is code that can be

written in C (or assembler).

Option A: 50 points. No disk I/O.

Option B: 60 points: linked list with disk storage

Option D: 70 points access list via sortable arrays of pointers

to nodes, and provide multiple sort views into the database from

the main menu.

Option : +10 points allow edit and delete of records

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=

Using the discussion in course materials, create an in-memory CIRCULAR

DOUBLY LINKED linked-list database program which displays a menu screen

similar to the following...

LINKED LIST PROGRAM

e) Add new agent to end of list

l) List all agents

n) Display Next record

p) Display Previous record

q) Quit program

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

Current Record:

0X567A:::0X620A:::0X640D James Bond 001

...and which processes SINGLE KEYSTROKE INPUT from the user.

On the Linux system, this is possible using the instructor's

/var2/local/include/getchne.h

code, on Microsoft, use the getch() function which should be in the

System.Console namespace now, or listed in conio.h.

If you use the conio.h library under Windows, Write the code so that

a single #define UNIX command allows the code to be compiled and run on

either system.

You can choose between STRUCT and CLASS record structures as follows:

struct prs

{

int useme ;

char name[30] ;

int mum ;

struct prs \* prev ;

struct prs \* next ;

} \*P, \*phere , \*pnew ;

OR:

class prs

{

public:

int useme ; // deleted?

char name[30] ;

int num ;

prs \* prev ; // prev node

prs \* next ; // next node

prs() ; // constructor

~prs() ; // destructor

} \*P, \*phere , \*pnew ;

(now, in a Truly Obsessively Compulsively Disordered OOP shop, you'd use:

class prs

{

private:

static int totalrecs ;

int useme ; // deleted?

char name[30] ;

int num ;

prs \* prev ; // prev node

prs \* next ; // next node

public:

prs() ; // constructor

~prs() ; // destructor

int getUseme(void) ;

void setUseme(int) ;

char \* getName(void) ;

void setName(char \*) ;

int getNum(void) ;

void setNum(int) ;

prs \* getPrev() ;

void setPrev(prs\*) ;

prs \* getNext() ;

void setNext(prs\*) ;

void edit\_rec(void) ;

} ;

...but this's for Christmas Vacation, let's try to keep focused on

linking the list.)

Access to the list will be via a prs \* "anchor" ; variable

which will serve as the access point to the list by pointing to the

first record. Thereafter, each record will be accessed via the preceeding

record. Each record should point "forward" to the next record and "back"

to the preceeding record.

SUGGESTIONS FOR FUNCTIONS:

prs::print\_rec(): display a record, including pointer contents.

pop\_menu() : clear screen, display menu, get choice, and return number of

choice to calling routine. In addition to displaying the menu choices,

it should display the "current" record in the list, that is, the one

phere points to.

listall(): calls print\_rec to display all records in list.

alloc\_rec() : creates a record

edit\_rec() : prompts the user for input and fills the record fields.

Design so that later on you can include extra functions.

Work for simplicity and clarity.

MAJOR HINTS: To do "display next record" or "display

previous record" all you have to do is: "phere =

phere->next" or "phere = phere->prev" and then re-display

the program screen. If you think about it, you ought to be

able to use the same logic in the "list all agents" option.

Symbolic Situation: three records linked into list (1,

2, and 3), it is just after allocating new record (7),

but before linking it into list ("num" field omitted

for space):

p1: &1 pnew: &7

&1 &2 &3 &7

name: one name: two name: three name: dOxk2

prev: &3 prev: &1 prev: &2 prev:^&(&43 (junk)

next: &2 next: &3 next: &1 next:/9Xik5 (junk)

Situation just after linking in the new record and

getting name:

p1: &1 pnew: &7

&1 &2 &3 &7

name: one name: two name: three name: four

prev: &7 prev: &1 prev: &2 prev: &3

next: &2 next: &3 next: &7 next: &1

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=

Create a program which does the following:

0. Manipulates Records as Class prs Objects.

(Class prs fields defined below:)

1. Loads Linked List from diskfile, if it exists. EITHER:

in one single block I/O operation, into an area of ram allocated precisely

to hold it as an array of structs. Once in RAM, step through the array

of structs, connecting the pointers in each record to the records ahead

and behind them, to produce a doubly-linked circular list (serial linkage,

both "next" and "prev" pointers correctly set)

OR: read file one record at a time, allocate each new record discretely,

and link it into the list. After file is read, close it, and display

menu for users.

If no file found, advises user and creates new database from user input.

New structs may then be added one at a time, with user input from

the keyboard. When the program ends, save all structs, one at a time,

to a binary disk file. The design of the struct is up to you, and you

may re-use as much of Assignment 6 as you like, to speed up the task.

LINKED LIST PROGRAM

e) Enter a new agent to end of list

n) Next record

p) Previous record

l) List agents in database

d) Delete agent file

q) Quit program

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

Current Record:

James Bond 001 Previous: 0X567A Current: 0X620A Next: 0X640D

2. Allows the user to

A.) Display the whole list to the screen,

B.) Add a record to the end of the list by keyboard input of fields contents.

C.) Display the next or previous struct in list.

D.) Quit program saving the list to disk.

3. Uses binary (unformatted fread or system-level) i/o.

PROGRAMMING SUGGESTIONS/HINTS: Modularize the program so

that reading in and reorganizing pointers in the list,

displaying a struct, writing a single struct to disk,

writing the array out, getting struct data from the

keyboard, are in separate procedures, all driven by a

main function built around a user menu. If you write this

program well, it will serve you as the basic example for

many useful programs you will want to develop for yourself

in years to come. Work for simplicity and clarity.

Extra Credit: Design program to include delete functions,

search functions, and edit functions. Extra Extra: use

dynalloced arrays of struct pointers to provide SORTED

"listall" access to list on NAME and NUMBER fields by using

a sorted array of pointers. (Download and run lldex binary

for example.)

Either alternative uses exactly the same record specs.

Sample data files are provided, which have records in

the form:

You may use eiher a:

struct prs

{

int useme ; // indicates record deleted

char name[30] ;

int num ;

prs \* pprev ; // previous rec in list

prs \* pnext ; // next rec in list

} or a:

class prs

{

int useme ; // indicates record deleted

char name[30] ;

int num ;

prs \* pprev ; // previous rec in list

prs \* pnext ; // next rec in list

prs() ; // constructor

~prs() ; // destructor

}

If you use the class, you might wish to write the constructor method

so that it links itself, so the external program need only handle the

pointers P->prev->next and P->prev; (Or not, it's up to you.)

(This assignment points to the subject material you will encounter in an

upper division class in Data Structures. Students who want to struggle

with this may email the instructor (tharrisb@sdccd.edu) for solutions to

these two programs after the semester's end.) If you don't see the

point, you will some day, when you're doing real work. Without this

kind of linked data structure, programs would have to use arrays,

and have no clean way of handling arbitrary numbers of duplicate input

lines, resulting in the loss of duplicate input data, or slow runtime

resolving collisions in an array. But with a linked data structure,

such complications evaporate.

Also, being able to arrange data in an organized way makes searching

databases very fast. The "B-Tree" is the foundation of modern high-speed

searching, and this assignment teaches all the tools needed to use one.

All contents copyright (C) T.E.H., all rights reserved. None of this

text may be displayed or stored on any public access or private access

system without the written consent of the author.

Full-on extra point mode accessing nodes via sorted pointer arrays:

(Arrays point to records, and each is sorted on a different field.

The nodes don't change positions in the list, that would be slow.

This is what is meant by "indexing" databases. In fact, this is the

same kind of code Ashton Tate used on the first dBase product, and

made millions.)

LINKED LIST PROGRAM

d (elete current record

e (nter new agent

l (ist all agents

m (odify current record

n (ext agent

p (revious agent

q (uit program

s (ort by name

# (ort by number

t (hrow away file

DESIGN CONSIDERATION:

There are TWO basic states of nature: P has no address in it, which

indicates that the user is creating the FIRST node, and P has an

address in it, indicating that a list already exists. Obviously, it's

crucial to initialize all your pointers when the program starts.

This kind of assignment really tells how much you know.

ALL THAT A PROGRAM "KNOWS" IS CONTAINED IN VARIABLES.

Use struct prs \* P as the "anchor".

Use struct prs phere as the "current record".

Use struct prs pnew as the "new record".