Assignment 4

Brandon Rubio C23727769 ECE421

FILE: “hw4.b” in /home/students/bur2/OS/Assignments/Assignment4

I was able to get the core of the program working in that the add(string, topOfList), remove(string,topOfList), and all() functions work properly and they can be seen in the following:

Text

Description automatically generated

Using this code in the start function:

top := newLink("test", nil);

add("one", top);

add("two", top);

add("three", top);

all(top);

out("\n");

top := remove("one", top);

all(top);

out("\n");

top := remove("test", top);

all(top);

out("\n");

top := remove("three", top);

all(top);

out("\n");

top := remove("two", top);

all(top);

However, I’m having trouble getting the interactive portion of the program to work properly. I keep finding different bugs that I think stem from not completely understanding the inch() function. I believe that I have to wait until the inch function cycles through every character that is inputted into the buffer otherwise inch() doesn’t wait for another user in put like I would like it to. I will modify my program as needed and will try to resubmit a full working version as soon as possible.

**Code:**

import "io"

manifest {linkSize = 2, stringInputSize = 300, data\_i = 0, next\_i = 1} // i = index

let strdup(string) be { //returns a vector of characters

let cpy = newvec(stringInputSize);

//double check this

for i = 0 to stringInputSize do {

cpy ! i := byte i of string;

if byte i of string = 0 then {

resultis cpy;

}

i +:= 1;

}

cpy ! stringInputSize := 0;

resultis cpy;

}

let strcmp (a, b) be {

let i = 0;

//out("entered strcmp, a = %s, b = %s\n", a, b);

while 1 do {

if byte i of a <> byte i of b then {

resultis false;

}

if byte i of a = 0 \/ byte i of b = 0 then {

break;

}

i +:= 1;

}

if byte i of a = 0 /\ byte i of b = 0 then {

resultis true;

}

resultis false;

}

let strncmp(a, b, max) be {

for i = 0 to max do

{ if byte i of a <> byte i of b then

resultis false;

if byte i of a = 0 then

resultis true }

resultis true

}

let readline() be {

let character, buffer = vec(stringInputSize + 1); // + 1 for null character

let index = 0;

for j = 0 to stringInputSize do {

character := inch();

if character = '\n' then {

break;

}

buffer ! index := character;

out("%d character: %c, index: %d\n", j, character, index);

if index <> stringInputSize do {

index +:= 1;

}

}

test index = stringInputSize then { // buffer became full

buffer ! (stringInputSize + 1) := '\0';

}

else { // character became '\n'

buffer ! index := '\0';

}

out("testing buffer, buffer = %s\n", buffer);

}

let printString(string) be {

out("%s\n", string);

}

let newLink(data, next) be {

//data is a pointer to a vector of characters (string), next is a pointer to another link

//data is the address of the string, must input an address for this to work

let r = newvec(linkSize);

r ! data\_i := data;

r ! next\_i := next;

resultis r;

}

let add(string, topOfList) be {

let ptr = topOfList;

while ((ptr ! next\_i) <> nil) do {

ptr := ptr ! next\_i;

}

ptr ! next\_i := newLink(string, nil);

resultis ptr ! next\_i;

}

let remove(string, topOfList) be { //returns top of list

let temp;

let ptr = topOfList;

let top = topOfList;

test strcmp(string, ptr ! data\_i) then {

temp := ptr ! next\_i;

freevec(ptr);

resultis temp;

}

else {

while ((ptr ! next\_i) <> nil) do {

test strcmp(string, (ptr ! next\_i) ! data\_i) then {

temp := ptr ! next\_i;

ptr ! next\_i := (ptr ! next\_i) ! next\_i;

freevec(temp);

break;

}

else {

ptr := ptr ! next\_i;

}

}

resultis top;

}

}

let all(topOfList) be {

let ptr = topOfList;

while (ptr <> nil) do {

printString(ptr ! data\_i);

ptr := ptr ! next\_i;

}

}

let clearBuffer(buf) be {

for i = 0 to stringInputSize do {

buf ! i := nil;

}

}

let start() be {

let heap = vec(10000);

let top, i = 1, buffer = vec(stringInputSize);

let result;

init(heap, 10000);

set\_kb\_buffer(buffer, stringInputSize);

out("Enter the strings and deletions:\n");

top := nil;

top := newLink("test", nil);

add("one", top);

add("two", top);

add("three", top);

all(top);

out("\n");

top := remove("one", top);

all(top);

out("\n");

top := remove("test", top);

all(top);

out("\n");

top := remove("three", top);

all(top);

out("\n");

top := remove("two", top);

all(top);

/\*

while 1 do {

inch();

//out("buffer = %s\n", buffer);

if strcmp(buffer, "ALL\n") then{

//out("if all\n");

all(top);

clearBuffer(buffer);

loop;

}

if strcmp(buffer, "DELETE\n") then{

out("Enter word to delete\n");

inch();

top := remove(buffer, top);

clearBuffer(buffer);

loop;

}

if strcmp(buffer, "EXIT\n") then {

//out("break\n");

clearBuffer(buffer);

break;

loop;

}

test i = 1 then {

//out("i = 1\n");

top := newLink(buffer, nil);

out("top data: %s, next: %d\n", top ! data\_i, top ! next\_i);

i := -1;

}

else {

//out("i != 1 \n");

if buffer ! 0 <> nil then {

out("added\n");

result := add(buffer, top);

//out("result: %d, data : %s, next: %d\n", result, result ! data\_i, result ! next\_i);

}

}

//out("before buffer size: %d\n", strlen(buffer));

clearBuffer(buffer);

//out("after buffer size: %d\n", strlen(buffer));

}

\*/

}