Assignment 4 Resubmission

Brandon Rubio C23727769 ECE421

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

I was able to fix my issues in the previous assignment, everything is working properly now.

**TEST RUN**

Enter the strings and deletions:

hello

this

program

works

ALL

works program this hello

properly

DELETE

Enter word to delete

program

dog

ALL

dog properly works this hello

cat

DELETE

Enter word to delete

dog

ALL

cat properly works this hello

END

**CODE**

import "io"

manifest {linkSize = 2, bufferSize = 100, data\_i = 0, next\_i = 1} // i = index

let strdup(string) be {

// string must have terminating null character

let stringSize = strlen(string);

let cpy;

test ~((stringSize + 1) rem 4) then {

cpy := newvec((stringSize + 1) / 4);

}

else {

cpy := newvec((stringSize + 1) / 4 + 1);

}

//out("strdup string = %s, size = %d\n", string, stringSize);

for i = 0 to stringSize do {

byte i of cpy := byte i of string;

}

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 strcpy(a, b) be {

// copy from a to b;

let i = 0;

while 1 do {

byte i of b := byte i of a;

if byte i of a = 0 then {

break;

}

i +:= 1;

}

}

let expandString(string) be {

// input string that was created with newvec

let r = newvec((strlen(string) + 1) / 4 + 1); // assumes that string is completly full and needs another word

strcpy(string, r);

freevec(string); // may cause memory leak if old string is not erased since it will be replaced with r

resultis r; // return same string inputed except with an extra word

}

let addCharToString(char, string) be {

// initial string should be created with newvec

let r, stringSize = strlen(string);

test ~((stringSize + 1) rem 4) then { // if full

r := expandString(string);

byte strlen(r) of r := char;

byte strlen(r) + 1 of r := '\0';

}

else {

// this is so that regardless of what option is chosen, this function will destroy string and return a replacement string

r := newvec((stringSize + 1) / 4 + 1);

strcpy(string, r);

freevec(string);

byte strlen(r) of r := char;

byte strlen(r) + 1 of r := '\0';

}

resultis r;

}

let readline(buffer) be {

let character, cpy = newvec(1);

for i = 0 to bufferSize - 1 do { // done 100 times since the default input buffer for inch() is set to 100 characters

character := inch();

//out("done %d times, character = %c\n", index, character);

if character = '\n' then {

break;

}

cpy := addCharToString(character, cpy); // using cpy instead of buffer since buffer might not be created with newvec, necessary for expandString() function

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

}

strcpy(cpy, buffer);

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

freevec(cpy);

}

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 newTop;

newTop := newLink(string, topOfList);

resultis newTop;

}

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 ! data\_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 ! data\_i);

freevec(temp);

break;

}

else {

ptr := ptr ! next\_i;

}

}

resultis top;

}

resultis top;

}

let all(topOfList) be {

let ptr = topOfList;

while (ptr <> nil) do {

out("%s ", ptr ! data\_i);

ptr := ptr ! next\_i;

}

out("\n");

}

let clearBuffer(buf) be {

for i = 0 to strlen(buf) - 1 do {

buf ! i := nil;

}

}

let start() be {

let heap = vec(10000);

let top = nil, i = 1, buffer = vec(bufferSize);

let result;

let one, two, three, four, five, yes;

init(heap, 10000);

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

while 1 do {

readline(buffer);

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

if strcmp(buffer, "ALL") then {

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

all(top);

loop;

}

if strcmp(buffer, "DELETE") then {

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

readline(buffer);

top := remove(buffer, top);

loop;

}

if strcmp(buffer, "END") then {

//out("break\n");

break;

loop;

}

if strlen(buffer) <> 0 then {

//out("added\n");

top := add(strdup(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));

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

}

}