Maria Martinez

Justin Swenson

04/22/2014

CS 3240

Program #1

// reverse.cpp

#include <iostream>

#include <new>

#include <stdlib.h>

using namespace std;

struct node {

char atom; // 0 or 1

char data; // if atom: actual data

node \*link; // ptr to list

node \*next; // ptr to next node

};

node \* const nill = (node \*) 0;

int const maxs = 80; // max string length

char const lefp = '(', newl = '\n', ritp = ')';

node \*stack[maxs >> 1], \*\*top = stack - 1; // max stack size <= maxs/2

int main()

{

node \*linklist( const char \* );

int getstring( char \* );

void newerr();

int ok( const char \* );

void deleteblanks( char \* );

void drawline( void );

void echoprint( const char \* );

void eraselist( node \* );

void revlist( const node \* );

void revstring( const char \* );

void scanlist( const node \* );

node \*head; // ptr to list structure

char s[maxs+1]; // data string; length <= maxs

set\_new\_handler( newerr );

while ( getstring( s ) ) {

echoprint( s );

deleteblanks( s );

if ( !ok( s ) ) {

cout << "\*\* illegal list format \*\*\n";

}

else {

revstring( s );

head = linklist( s );

scanlist( head );

revlist( head );

eraselist( head );

}

}

drawline();

return 0;

}

void deleteblanks( char \*s ) {

char \*p, \*q;

cout << "Deblanked string:\n";

for ( p = q = s ; \*q ; q++ ) {

if ( ( \*q != ' ' ) && ( \*q != '\t') ) { // if \*q is not empty

\*p++ = \*q; // increment p to point to q

}

}

\*p = '\0';

cout << s << newl;

}

void drawline() {

int i;

for ( i = 75 ; i -- ;) // print n times

cout << "-" ;

cout << newl;

}

void echoprint( const char \*s ) {

cout << "Echo of data string:\n" << s << newl; // print user input

}

void eraselist( node \*p ) {

if ( p != nill ) { // if p is not null

eraselist( p->next ); // erase p->next

if ( !p->atom ) // if p is not an atom

eraselist( p->link ); // erase p->link

delete p;

}

}

int getstring( char \*s ) {

drawline();

cout << "Type a string representing a generalized list, please:\n";

cin.getline( s, maxs+1 ); // read 80 chars and convert new line to null terminator

return( (int)\*s ); // return integer value of \*s

}

node \* linklist( const char \*s ) {

node \*newnode( void );

void pop( node \*& ), push( node \* );

node \*p = nill, \*q;

int lp = 0;

char ch;

while ( \*s ) { // while s is true

ch = \*s++; // current char is \*s then increment s to point to next char

if ( ch == ritp ) { // if current char is a right paran

lp = 0; // then it's the end of a link

pop( p ); // return to last left paran ( since link is closed by right paran )

}

else {

q = newnode(); // q is a new node

if ( p ) { // if p is true

if ( lp ) // if p is a left paran

p->link = q; // then p's link is new node q

else // p is not a left paran

p->next = q; // then p's next is new node q

}

p = q; // p is q

lp = ( ch == lefp ); // left param true if curent char is left paran

q->atom = (char)!lp; // q is atom if char is not a left paran

if ( q->atom ) // if q is atom

q->data = ch; // atom data is current char

else // then q is left paran

push( q ); // push this to the stack to return to later

}

}

return p;

}

void newerr() {

cout << "\*\* can't allocate space for node \*\*\n";

exit( 1 );

}

node \* newnode() {

node \*q;

q = new node; // create new node

q->link = q->next = nill; // initialize node's link and node's next to nill

return( q );

}

int ok( const char \*s ) {

int n = 0;

if ( \*s != lefp ) { // if string does not begin with left paran

return 0; // return false

}

else {

n = 0; // initialize n to 0

do {

switch ( \*s ) { // switch case

case lefp: n++; // if left paran then increment n

break;

case ritp: --n; // if right paran then decrement n

break;

default: // default is blank

break;

}

} while ( \*++s && 0 < n ); // do while s points to something and n > 0

}

return( (\*s == '\0') && ( n == 0 ) ); // legal string if s is null terminator and n is 0

}

void pop( node \*& p ) {

p = \*top--; // set p to next item on top of stack

}

void push( node \*p ) {

\*++top = p; // push p to top of stack

}

void rev( const node \*p ) {

if ( p != nill ){ // if p is not nill

rev( p->next ); // rev p's next

if ( p->atom ) // if p is an atom

cout << p->data; // print atom's data

else { // p is not an atom

cout << lefp; // print left paran

rev( p->link ); // rev p's link

cout << ritp; // print right paran

}

}

}

void revlist( const node \*p ) {

cout << "Reversed linked list:\n";

rev( p );

cout << newl;

}

void revstring( const char \*s ) {

const char \*p;

char ch;

cout << "Reversed string:\n";

p = s;

while ( \*p ) {

p++;

}

while ( s < p ) {

switch ( ch = \*--p ) {

case lefp: ch = ritp;

break;

case ritp: ch = lefp;

break;

default:

break;

}

cout << ch;

}

cout << newl;

}

void scan( const node \*p ) {

if ( p != nill ) {

if ( p->atom )

cout << p->data;

else {

cout << lefp;

scan( p->link );

cout << ritp;

}

scan( p->next );

}

}

void scanlist( const node \*p ) {

cout << "Scan of linked list:\n";

scan( p );

cout << newl;

}

RESULTS:

**---------------------------------------------------------------------------**

**Type a string representing a generalized list, please:**

(the () fox () jumps () over () the () lazy () dog () )

**Echo of data string:**

**(the () fox () jumps () over () the () lazy () dog () )**

**Deblanked string:**

**(the()fox()jumps()over()the()lazy()dog())**

**Reversed string:**

**(()god()yzal()eht()revo()spmuj()xof()eht)**

**Scan of linked list:**

**(the()fox()jumps()over()the()lazy()dog())**

**Reversed linked list:**

**(()god()yzal()eht()revo()spmuj()xof()eht)**

**---------------------------------------------------------------------------**

**Type a string representing a generalized list, please:**

(my dog's name is ( puka (she is a boston ( terrier ) ) ) )

**Echo of data string:**

**(my dog's name is ( puka (she is a boston ( terrier ) ) ) )**

**Deblanked string:**

**(mydog'snameis(puka(sheisaboston(terrier))))**

**Reversed string:**

**((((reirret)notsobasiehs)akup)siemans'godym)**

**Scan of linked list:**

**(mydog'snameis(puka(sheisaboston(terrier))))**

**Reversed linked list:**

**((((reirret)notsobasiehs)akup)siemans'godym)**

**---------------------------------------------------------------------------**

**Type a string representing a generalized list, please:**

(uh oh (( this won't work )

**Echo of data string:**

**(uh oh (( this won't work )**

**Deblanked string:**

**(uhoh((thiswon'twork)**

**\*\* illegal list format \*\***

**---------------------------------------------------------------------------**

**Type a string representing a generalized list, please:**

**---------------------------------------------------------------------------**

Program ended with exit code: 0