#include <stdio.h>

#include <stdlib.h>

#define length 7

#define Maxlength 100

const char\* action[] = {"Start", "LEFT 1", "LEFT 2", "RIGHT 1", "RIGHT 2"};

typedef struct {

int data[length];

int vt;

}State;

typedef struct {

State state;

struct Node\* Parent;

int no\_function;

}Node;

typedef struct {

Node\* Element[Maxlength];

int Top\_idx;

}Stack;

void push(Node\* x, Stack \*stack){

if(full\_Stack(\*stack)){

printf("Error! Stack is full");

} else {

stack->Top\_idx--;

stack->Element[stack->Top\_idx] = x;

}

}

void makeNull\_Stack(Stack \*stack){

stack->Top\_idx = Maxlength;

}

int empty\_Stack(Stack stack){

return stack.Top\_idx == Maxlength;

}

Node\* top(Stack stack){

if(!empty\_Stack(stack)){

return stack.Element[stack.Top\_idx];

}

return NULL;

}

void pop(Stack \*stack){

if(!empty\_Stack(\*stack)){

stack->Top\_idx++;

} else {

printf("Error! Stack is empty");

}

}

int full\_Stack(Stack stack){

return stack.Top\_idx==0;

}

void makeNullState(State \*state){

int i;

for(i=0; i<length; i++){

state->data[i] =0;

}

state->vt=3;

}

int left1(State cur, State \*re){

if(cur.vt > 0 && cur.data[cur.vt-1] == 1){

re->data[cur.vt] = 1;

re->data[cur.vt-1] = 0;

re->vt = cur.vt -1;

return 1;

}

return 0;

}

int left2(State cur, State \*re){

if(cur.vt > 1 && cur.data[cur.vt-1] == 2 && cur.data[cur.vt-2] == 1){

re->data[cur.vt] = 1;

re->data[cur.vt-2] = 0;

re->vt = cur.vt -2;

return 1;

}

return 0;

}

int right1(State cur, State \*re){

if(cur.vt < 6 && cur.data[cur.vt+1] == 2){

re->data[cur.vt] = 2;

re->data[cur.vt+1] = 0;

re->vt = cur.vt +1;

return 1;

}

return 0;

}

int right2(State cur, State \*re){

if(cur.vt < 5 && cur.data[cur.vt+1] == 1 && cur.data[cur.vt+2] == 2){

re->data[cur.vt] = 2;

re->data[cur.vt+2] = 0;

re->vt = cur.vt +2;

return 1;

}

return 0;

}

int call\_operator(State cur, State \*re, int opt){

switch(opt){

case 1: return left1(cur, re);

case 2: return left2(cur, re);

case 3: return right1(cur, re);

case 4: return right2(cur, re);

default:

printf("Error call\_operator");

return 0;

}

}

int goalCheck(State state){

if(state.data[0] == 2 && state.data[1] == 2 && state.data[2] == 2 && state.data[3] == 0 && state.data[4] == 1 && state.data[5] == 1 && state.data[6] == 1){

return 1;

}

return 0;

}

int compareState(State s1, State s2){

int i;

for(i=0; i<7; i++){

if(s1.data[i] != s2.data[i]){

return 0;

}

}

return 1;

}

int findState(State state, Stack S){

while(!empty\_Stack(S)){

Node\* node = top(S);

if(compareState(node->state, state)){

return 1;

}

pop(&S);

}

return 0;

}

void printState(State state){

int i;

for(i=0; i<7; i++){

printf("%d ", state.data[i]);

}

printf("vt=%d\n", state.vt);

}

Node\* DFS(State state){

Stack open;

Stack close;

makeNull\_Stack(&open);

makeNull\_Stack(&close);

Node\* root;

root->no\_function = 0;

root->Parent=NULL;

root->state=state;

push(root, &open);

while(!empty\_Stack(open)){

Node\* node = top(open);

push(node, &close);

pop(&open);

if(goalCheck(node->state)){

return node;

}

int opt;

for(opt=1; opt<=4; opt++){

State newState;

makeNullState(&newState);

int j;

for(j=0; j<7; j++){

newState.data[j] = node->state.data[j];

}

if(call\_operator(node->state, &newState, opt)){

if(findState(newState, open) || findState(newState, close)){

continue;

}

Node\* newNode = (Node\*)malloc(sizeof(Node));

newNode->no\_function = opt;

newNode->Parent = node;

newNode->state = newState;

push(newNode, &open);

}

}

}

return NULL;

}

void printGoal(Node\* p){

Node\* List[Maxlength];

int count=0;

while(p->Parent != NULL){

List[count] = p;

count++;

p=p->Parent;

}

List[count] = p;

count++;

p=p->Parent;

int i;

for(i = count-1; i>=0; i--){

Node\* node = List[i];

printf("action %d: %s\n", node->no\_function, action[node->no\_function]);

printState(node->state);

}

}

int main(){

State state;

state.data[0] = 1;

state.data[1] = 1;

state.data[2] = 1;

state.data[3] = 0;

state.data[4] = 2;

state.data[5] = 2;

state.data[6] = 2;

state.vt = 3;

Node\* p = DFS(state);

printGoal(p);

return 0;

}