```
#include <fstream>
#include <cstdlib>
#include <ctime>
#include <string>
#include <iostream>

//Game Architecture Types and Headers
#include "GUI.h"
#include "Timer.h"
#include "engine.h"
#include "constants.h"

//Game Specific Types
#include "Link.h"
#include "Block.h"

using namespace std;

int main(int argc, char *argv[]){


    //Initialize GUI->Engine communication*/
    LK_TRANSITION command = NA;

    //Construct Timer
    Timer fps;

    //Construct GUI
    GUI gui;

    //Construct link
    Link* link = new Link;

    //Construct Blocks
    int numBlocks = 0;
    Block** blocks = constructBlocks("./Assets/Config/level.txt",numBlocks);

    //Initialize Termination Criteria
    bool quit = false;


    //While the user hasn't quit
    while(quit == false){
        //Start the frame timer
        fps.start();

        //GUI waits for mouse events
        while(gui.observeEvent()){

            //Receive Command From GUI
            command = gui.getCommand();

            //GUI transmits quit event
            if(gui.quitGame()){

                quit=true;

            }

        }

        //Task 3: Update Game state
        changeGameState(command, link, blocks, numBlocks);

        //Render Game Data
        gui.displayGameState(link,blocks,numBlocks);

        //Adjust Frame Rate
        if(fps.get_ticks() < FRAME_DELAY){
            SDL_Delay( FRAME_DELAY - fps.get_ticks() );
        }

    }

    //Return
    return 0;

}
```

```cpp
#ifndef ENGINE_H
#define ENGINE_H

#include <string>

#include "constants.h"
#include "Link.h"
#include "Block.h"

Block** constructBlocks(string,int&);
void changeGameState(LK_TRANSITION, Link*, Block**, int);

#endif
```

```
#include <fstream>
#include <iostream>
#include <string>

//Game Architecture Headers
#include "engine.h"

using namespace std;

Block** constructBlocks(string file, int & numBlocks){

    fstream fin;
    fin.open(file.c_str(),ios::in);

    //Get Number of Blocks in configuration file
    fin >> numBlocks;

    //Construct each block and store it in the array
    Block** blocks = new Block*[numBlocks];

    //Read in the blocks
    float blockX=0.0f,blockY=0.0f;
    for(int i=0;i<numBlocks;i++){
        fin >> blockX;
        fin >> blockY;
        blocks[i] = new Block(blockX,blockY);
    }

    //Clean-up
    fin.close();

    return(blocks);

}

void changeGameState(LK_TRANSITION command, Link* link, Block** blocks, int numBlocks){

    //Update Link
    link->update(command);

    //Update each block individually
    for(int i=0;i<numBlocks;i++){
        blocks[i]->update(link->getPosX());
    }

    //Boundary Checking (scroll boundary)
    if(link->getPosX() > SCREEN_WIDTH/2){

        link->setPosX(SCREEN_WIDTH/2);

        //Update each block individually
        for(int i=0;i<numBlocks;i++){
            blocks[i]->setPosX(blocks[i]->getPosX()-LK_RUN_SPEED);
        }
    }

}
```

```
#ifndef LINK_H
#define LINK_H

#include <string>
#include "constants.h"

using namespace std;

class Link{

public:

    Link();
    ~Link();

    //Get Methods
    int getPosX() const;
    int getPosY() const;
    int getState() const;
    int getSpriteID() const;

    //Set Methods
    void setPosX(float);
    void setPosY(float);
    void setState(int);
    void setSpriteID(int);

    //Animation Function
    void update(LK_TRANSITION);

private:

    //Declare  Link Properties
    float posX;
    float posY;
    int state;
    int spriteID;
    int animationID;

    //Storage
    int numStates;
    int* animationSize;
    int** animationMap;

    //Private functions to manipulate internal Link state
    void loadAnimation(string);
    void updateSprite();
    void moveDown();
    void moveAttack();
    void moveLeft();
    void moveRight();
    void noAction();

};
```

```
#endif
```

```cpp
#include <iostream>
#include <fstream>

//Game Architecture Headers
#include "constants.h"
#include "Link.h"

Link::Link(){

    //Initialize at bottom left of screen
    posX = 0.0f;
    posY = (float)(SCREEN_HEIGHT-LK_SPRITE_HEIGHT);

    //Standing, facing to the right
    state = STILL_RIGHT;
    animationID = 0;

    //Load Animation Data
    loadAnimation("./Assets/Config/animation.txt");
    //Compute Initial Sprite
    updateSprite();

}

Link::~Link(){

    //Clean-up ragged 2D array
    for(int i=0;i<numStates;i++){
        delete [] animationMap[i];
    }
    delete [] animationMap;

    //Clean-up 1D array
    delete [] animationSize;

}

void Link::loadAnimation(string gameFile){

    //Declare and open filestream
    fstream fin;
    fin.open(gameFile.c_str(),ios::in);

    //Number of columns to store
    fin >> numStates;

    //Allocate memory
    animationSize = new int[numStates];
    animationMap = new int*[numStates];

    //Load the ragged array
    for(int i=0;i<numStates;i++){

        fin >> animationSize[i];
        animationMap[i] = new int[animationSize[i]];

        for(int j=0;j<animationSize[i];j++){
            fin >> animationMap[i][j];
        }

    }

    //Clean-up
    fin.close();

}



int Link::getState() const{
    return(state);
}

int Link::getSpriteID() const{
    return(spriteID);
}

//Set Methods
void Link::setPosX(float posX){
    this->posX = posX;
}

void Link::setPosY(float posY){
    this->posY = posY;
}

void Link::setState(int state){
    this->state = state;
}

void Link::setSpriteID(int spriteID){
    this->spriteID = spriteID;
}



//Plotting Methods
int Link::getPosX() const{
    return((int)posX);
}

int Link::getPosY() const{
    return((int)posY);
}

void Link::update(LK_TRANSITION command){

    switch(command){
        //Execute the appropriate state transition
        case ATTACK: moveAttack(); break;
        case DOWN: moveDown(); break;
        case LEFT: moveLeft(); break;
        case RIGHT: moveRight(); break;
        case NA: noAction(); break;
    }

    //Link Boundary Checking (left boundary)
    if(getPosX() <= 0){
        setPosX(0);
    }

    //Boundary Checking (top and bottom boundaries)
    if(getPosY() >= SCREEN_HEIGHT-LK_SPRITE_HEIGHT){
        setPosY((float)(SCREEN_HEIGHT-LK_SPRITE_HEIGHT));
    }
    if(getPosY() <= 0){
        setPosY(0.0f);
    }

}

void Link::moveAttack(){

    //Changed states: initialize this state
    switch (state)
    {
    case STILL_RIGHT:
        state = ATTACK_RIGHT;
        animationID = 0;
        break;
    case STILL_LEFT:
        state = ATTACK_LEFT;
        animationID = 0;
        break;
    default:
        break;
    }

    updateSprite();
}
void Link::moveDown()
{
    switch (state)
    {
    case STILL_LEFT:
        state = CROUCH_LEFT;
        animationID = 0;
        break;
    case STILL_RIGHT:
        state = CROUCH_RIGHT;
        animationID = 0;
        break;
    }
    updateSprite();
```

```
}


void Link::moveRight(){

    //Conduct the appropriate state transition and/or animation
    switch(state){
    case STILL_RIGHT:
        state = WALK_RIGHT;
        animationID = 0;
        posX += LK_RUN_SPEED;
        break;
    case WALK_RIGHT:
        posX += LK_RUN_SPEED;
        break;
    default:
        state=STILL_RIGHT;
        animationID = 0;
    }
    updateSprite();
}


void Link::moveLeft(){

    //Conduct the appropriate state transition and/or animation
    switch(state){
    case STILL_LEFT:
        state = WALK_LEFT;
        animationID = 0;

        posX -= LK_RUN_SPEED;
        break;
    case WALK_LEFT:

        posX -= LK_RUN_SPEED;
        break;
    default:
        state=STILL_LEFT;
        animationID = 0;
    }
    updateSprite();

}

void Link::noAction(){

    if (state != STILL_LEFT || state != STILL_RIGHT)
    {
        //Conduct the appropriate state transition
        switch (state) {
        case ATTACK_RIGHT:
        case CROUCH_RIGHT:
        case WALK_RIGHT:
            state = STILL_RIGHT;
            break;
        case ATTACK_LEFT:
        case CROUCH_LEFT:
        case WALK_LEFT:
            state = STILL_LEFT;
            break;
        }
        //Reset animation and update the sprite
        animationID=0;
        updateSprite();
    }

}

void Link::updateSprite(){

    animationID++;

    //Wrap animation sequence
    if(animationID>=animationSize[state]){
        animationID = 0;
    }

    //Map sprite ID
    spriteID = animationMap[state][animationID];
}
```

```
#ifndef BLOCK_H
#define BLOCK_H

#include <string>
using namespace std;

class Block{

public:

    Block(float,float);

    //Accessor methods
    int getPosX();
    int getPosY();
    bool getReached();

    void setPosX(float);
    void setPosY(float);
    void setReached(bool);

    //Action methods
    void update(float);

private:

    //Members
    float posX;
    float posY;
    float velY;
    bool reached;

};

#endif
```

```cpp
#include <cmath>
#include <iostream>

#include "constants.h"
#include "Block.h"

using namespace std;

Block::Block(float posX, float posY){
    this->posX = posX;
    this->posY = posY;
    reached = false;
}

bool Block::getReached(){
    return(reached);
}

void Block::setPosX(float posX){
    this->posX = posX;
}

void Block::setPosY(float posY){
    this->posY = posY;
}

void Block::setReached(bool reached){
    this->reached = reached;
}

int Block::getPosX(){
    return((int)posX);
}

int Block::getPosY(){
    return((int)posY);
}


void Block::update(float posPlayerX){

    if(posPlayerX > posX){
        reached = true;
    }

    if(reached){
        posY += DELTA_T*velY;
        velY += DELTA_T*GRAVITY;
    }

}
```