CSc 256 Chapter 4 Assignment

An ancient robot game

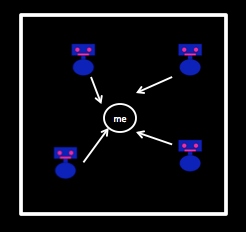
Due 11:59pm Sunday 12/13/2020

(15% of your grade)

For this project, you are given a program that implements a game where a human user tries to escape from four robots. The user and four robots are on an x-y grid. On every step, you enter a move for the human. The robots will attempt to get closer to the human.

When a robot has the same x-y coordinates as the human, the game is over. You will translate this program faithfully, following all function call guidelines and MIPS register use conventions.

Two arrays x[4] and y[4] keep track of the x- and y-coordinates of four robots. The positions of the human and the four robots are initialized in the program. On each step, the user enters a move; the positions of the human and the robots are updated. This continues until the human dies. This figure gives an idea of the game; we’ll work with a text-only version, sorry.



In the main loop, the user is prompted to enter a move. The position of the user is updated. Then the program calls a function **moveRobots()** to update the position of the robots as they try to catch the human. The new positions of the human and the robots are then displayed.

The function **moveRobots()** has prototype

int moveRobots(int \*arg0, int \*arg1, int arg2, int arg3)

arg0 is the base address of array that contains the x-coordinates of the four robots, arg1 is the base address of array that contains the y-coordinates of the four robots, arg2 is the x-coordinate of the human, arg3 is the y-coordinate of the human. **moveRobots()**updates the positions of the four robots, and returns a 1 if the human is alive, and a 0 if the human is dead (i.e., the human has the same coordinates as a robot). Each coordinate of a robot is updated by calling the function **getNew()**, which returns the new coordinate based on the current coordinate of the robot and the current coordinate of the human.

When you translate **moveRobots()** to MIPS assembly language, arg0 through arg3 are in $a0 through $a3; the return value is in $v0.

The function **getNew()** uses simple rules to move a robot closer to the human. If the difference in the coordinates is >=10, the robot's coordinate will move 10 units closer to the human. If the difference in the coordinates is < 10, the robot's coordinate will

move one unit closer to the human. (See program listings.) **getNew()** has prototype

**int getNew(int arg0, int arg1)**

arg0 is the coordinate (x or y) of a robot, arg1 is the coordinate (x or y) of the human. **getNew()** returns the new coordinate of the robot, based on the position of the human.

The function **getNew()** is already translated to MIPS assembly language, arg0 and arg1 are in $a0 and $a1 respectively, and the return value is in $v0.

A copy of the C++ program **robots.cpp** can be found on iLearn. The file **robots.asm** contains the main program and getnew(), already translated into MIPS assembly language. Your functions will follows the main program in the same file.

**Write the functions exactly as described in this handout.** Do not implement the program using other algorithms or tricks. Do not even switch the order of the arguments in function calls; **you must follow the order specified in the C++ code**. The purpose of this program is to test whether you understand nested functions. If you wish to make changes to the algorithm, you must first check with the instructor.

Your functions should be properly commented. Each function must have its own header block, including the prototype of the function, the locations of all arguments and return values, descriptions of the arguments and how they are passed, and a description of what the function does. Paste in the C++ code as inline comments for your MIPS assembly code. Refer to the Programming Style handout from Chapter 2 slides from iLearn for guidelines on how to comment your code.

You should try to make your code efficient. For example, your loops should follow the efficient form presented in class. Points will be deducted for obvious inefficiencies.

**Submission:** submit your code via iLearn. All your code should be in a single plain text file.

80% of your grade is for correctness. 20% is for clarity/documentation.

#include <iostream>  
int moveRobots(int \*, int \*, int, int );  
int getNew(int, int);  
  
int main()  
{  
 int x[4], y[4], i, j, myX = 25, myY = 25, move, status = 1;  
  
 // initialize positions of four robots  
 x[0] = 0; y[0] = 0;  
 x[1] = 0; y[1] = 50;  
 x[2] = 50; y[2] = 0;  
 x[3] = 50; y[3] = 50;  
  
 std::cout << "Your coordinates: 25 25\n";  
  
 while (status == 1) {  
 std::cout << "Enter move (1 for +x, -1 for -x, 2 for + y, -2 for -y):";  
 std::cin >> move;  
  
 // process user's move  
 if (move == 1)  
 myX++;  
 else if (move == -1)  
 myX--;  
 else if (move == 2)  
 myY++;  
 else if (move == -2)  
 myY--;  
  
 // update robot positions  
 status = moveRobots(&x[0],&y[0],myX,myY);  
  
 std::cout << "Your coordinates: " << myX << " " << myY <<std:: endl;  
  
 for (i=0;i<4;i++)  
 std::cout << "Robot at " << x[i] << " " << y[i] << std::endl;  
  
 }  
 std::cout << "AAAARRRRGHHHHH... Game over\n";  
}  
  
  
int moveRobots(int \*arg0, int \*arg1, int arg2, int arg3)  
{  
 int i, \*ptrX, \*ptrY, alive = 1;  
  
 ptrX = arg0;  
 ptrY = arg1;  
  
 for (i=0;i<4;i++) {  
 \*ptrX = getNew(\*ptrX,arg2); // update x-coordinate of robot i  
 \*ptrY = getNew(\*ptrY,arg3); // update y-coordinate of robot i  
  
 // check if robot caught user  
 if ((\*ptrX == arg2) && (\*ptrY == arg3)) {  
 alive = 0;  
 break;  
 }  
 ptrX++;  
 ptrY++;  
 }  
 return alive;  
}  
  
// move coordinate of robot closer to coordinate of user  
int getNew(int arg0, int arg1)  
{  
 int temp, result;  
  
 temp = arg0 - arg1;  
 if (temp >= 10)  
 result = arg0 - 10;  
 else if (temp > 0)  
 result = arg0 - 1;  
 else if (temp == 0)  
 result = arg0;  
 else if (temp > -10)  
 result = arg0 + 1;  
 else if (temp <= -10)  
 result = arg0 + 10;  
  
 return result;  
}

Output%

Your coordinates: 25 25

Enter move (1 for +x, -1 for -x, 2 for + y, -2 for -y):**2**

Your coordinates: 25 26

Robot at 10 10

Robot at 10 40

Robot at 40 10

Robot at 40 40

Enter move (1 for +x, -1 for -x, 2 for + y, -2 for -y):**1**

Your coordinates: 26 26

Robot at 20 20

Robot at 20 30

Robot at 30 20

Robot at 30 30

Enter move (1 for +x, -1 for -x, 2 for + y, -2 for -y):**2**

Your coordinates: 26 27

Robot at 21 21

Robot at 21 29

Robot at 29 21

Robot at 29 29

Enter move (1 for +x, -1 for -x, 2 for + y, -2 for -y):**1**

Your coordinates: 27 27

Robot at 22 22

Robot at 22 28

Robot at 28 22

Robot at 28 28

Enter move (1 for +x, -1 for -x, 2 for + y, -2 for -y):**2**

Your coordinates: 27 28

Robot at 23 23

Robot at 23 28

Robot at 27 23

Robot at 27 28

AAAARRRRGHHHHH... Game over

libra%