Computer Architecture

Project 2

Nolan Gregory

17 April 2023

Contents

C Algorithm

```c
#include <stdlib.h>
#include <stdio.h>
#include <time.h>

int find(int* arr, int num, int count);
int findBalls(int* generated, int* input);
int findStrikes(int* generated, int* input);
int generate(int* generated, int count);
int getInput(int* input, int count);

const int count = 3;

int find(int* arr, int num, int count){
    for(int i=0; i<count; i++){
        if(arr[i] == num){
            return i;
        }
    }
    return -1;
}

int findBalls(int* generate, int* input){
    int index;
    int num;
    int balls = 0;
    for(int i=0; i<count; i++){
        num = generate[i];
        index = find(input, num, count);
        if(index != -1 && index != i){
            balls++;
        }
    }
    return balls;
}

int findStrikes(int* generate, int* input){
    int index;
    int num;
    int strikes = 0;
    for(int i=0; i<count; i++){
        num = generate[i];
        index = find(input, num, count);
        if(index == i){
            strikes++;
        }
    }
    return strikes;
}

int generate(int* generated, int count){
    int randomNumber;
    int index;
    int added = 0;
    while(added < count){
        srand(time(NULL));
        randomNumber = (rand() % 9) + 1;
        index = find(generated, randomNumber, count);
        if(index == -1){
            generated[added] = randomNumber;
            added++;
        }
    }
    return 0;
}

int getInput(int* input, int count){
    int userInput;
    int index;
    int added = 0;
    while(added < count){
        printf("Enter a value between (1-9): ");
        scanf("%d", &userInput);
        fflush(stdin);
        while(userInput > 9 || userInput < 1){
            printf("Invalid input. Must be between (1-9): \n");
            scanf("%d", &userInput);
            fflush(stdin);
        }
        index = find(input, userInput, count);
        if(index == -1){
            input[added] = userInput;
            added++;
        }
        else{
            printf("Must enter non-duplicate value.\n");
        }
    }
    return 0;
}
```

```
void clearBuffer(void){
  while ( getchar() != '\n' );
}

void print(int* generate, int* input){
    for(int i=0; i<count; i++){
        printf("Generated: %d  --  Input: %d\n", generate[i], input[i]);
    }
}


int main(void){
    int generated[3] = {0};
    int balls;
    int strikes;
    char choice;
    int deez;

    generate(generated, count);

    while(1){
        int input[3] = {0};
        getInput(input, count);
        balls = findBalls(generated, input);
        strikes = findStrikes(generated, input);
        print(generated, input);
        (strikes > 0 ? (strikes==1) ? printf("%d strike ", strikes) : printf("%d strikes ", strikes) : printf(""));
        (balls > 0 ? (balls==1) ? printf("%d ball ", balls) : printf("%d balls ", balls) : printf(""));
        if(balls == 0 & strikes == 0) printf("Out");
        printf("\n");
        if(strikes == count){
            printf("Good Job! Do you want to play again (y/n)?: ");
            clearBuffer();
            choice = getchar();
            while(choice != 'y' && choice != 'n'){
                printf("Invalid Input!\n");
                printf("Good Job! Do you want to play again (y/n)?: ");
                choice = getchar();
                clearBuffer();
            }
            if(choice == 'n') break;
            generate(generated, count);
        }
    }
    return 0;
}
```

:: end program ::

C++ Algorithm

```
#include <iostream>
#include <vector>
#include <string>
#include <random>
#include <time.h>

using std::cout;
using std::cin;
using std::vector;
using std::string;

const int count = 3;

int find(vector<int>& arr, int num, int count) {
    for(int i=0; i<count; i++){
        if(arr[i] == num){
            return i;
        }
    }
    return -1;
}

int generate(vector<int>& generated, int count){
    int randomNumber;
    int size = 0;
    while(size < count){
        srand(time(NULL));
        randomNumber = (rand() % 10) + 1;
        if (find(generated, randomNumber, count) == -1){
            generated[size] = randomNumber;
            size++;
        }
    }
    return 0;
}

int getUserInput(vector<int>& input, int count){
    int userChoice;
    int size=0;
    while(size < count){
        cout << "Enter user input: ";
        cin >> userChoice;
        while(userChoice > 9 || userChoice < 1){
            cout << "Enter a value between [1-9]: ";
            cin >> userChoice;
        }
        if (find(input, userChoice, count) == -1){
            input[size] = userChoice;
            size++;
        }
        else cout << "Must enter non-duplicate value.\n";
    }
    return 0;
}

int findBalls(vector<int>& generated, vector<int>& input){
    int balls = 0;
    int currentGen = 0;
    int idx = -1;
    for(int i=0; i<count; i++){
        currentGen = generated[i];
        idx = find(input, currentGen, count);
        if(idx != -1 && idx != i){
            balls++;
        }
    }
    return balls;
}

int findStrikes(vector<int>& generated, vector<int>& input){
    int strikes = 0;
    int currentGen = 0;
    int idx = -1;
    for(int i=0; i<count; i++){
        currentGen = generated[i];
        idx = find(input, currentGen, count);
        if(idx == i){
            strikes++;
        }
    }
    return strikes;
}

int main(){
    int balls;
    int strikes;
    string choice;
    vector<int> g(count, 0);
```

```
    generate(g, count);

    while(true){
        vector<int> userInput(count, 0);
        getUserInput(userInput, count);

        balls = findBalls(g, userInput);
        strikes = findStrikes(g, userInput);

        if(strikes > 0) cout << strikes << " strikes ";
        if(balls > 0) cout << balls << " balls ";
        if(balls == 0 && strikes == 0) cout << "Out";
        cout << std::endl;

        if(strikes == count){
            cout << "\nGood Job! Do you want to play again (y/n)?: ";
            cin >> choice;
            while(choice != "y" && choice != "n"){
                cout << "Invalid input!\n";
                cout << "Good Job! Do you want to play again (y/n)?: ";
                cin >> choice;
            }
            if(choice == "n") break;
            else{
                srand(time(NULL));
                vector<int> g(count, 0);
                generate(g, count);
            }
        }
    }
    return 0;
}
```

:: end program ::

Python Algorithm

```
import random

count = 3

def find(arr, num, count):
    for i in range(len(arr)):
        if arr[i] == num:
            return i
    return -1

def generate(generated, count=count):
    size = 0
    while size < count:
        randomNumber = random.randint(1, 9)
        generated[size] = randomNumber
        size += 1
    return generated

def getUserInput(arr, count=count):
    size = 0
    while size < count:
        userchoice = int(input("Enter a value between 1-9: "))
        found = find(arr, userchoice, count) == -1
        if found:
            arr[size] = userchoice
            size += 1
        else:
            print("Must enter non-duplicate value.")
    return arr

def findBalls(generated, arr):
    balls = 0
    for i in range(len(generated)):
        current = generated[i]
        index = find(arr, current, count)
        if index != -1 and index != i:
            balls += 1
    return balls

def findStrikes(generated, arr):
    strikes = 0
    for i in range(len(generated)):
        current = generated[i]
        index = find(arr, current, count)
        if index == i:
            strikes += 1
    return strikes

def main():
    generated = [0]*count
    generated = generate(generated)
    while True:
        userinput = [0]*count
        userinput = getUserInput(userinput)
        balls = findBalls(generated, userinput)
        strikes = findStrikes(generated, userinput)

        if strikes > 0:
            print(f"{strikes} strike(s)")
        if balls > 0:
            print(f"{balls} ball(s)")
        if balls == 0 and strikes == 0:
            print("Out")

        if strikes == count:
            choice = input ("Good Job! Do you want to play again (y/n)?: ").lower()
            while choice != "y" and choice != "n":
                print("Invalid input!")
                choice = input ("Do you want to play again (y/n)?: ").lower()
            if choice == "n":
                break
            generated = generate()
    return 0

if __name__ == "__main__":
    main()
```

:: end Frequency ::

MIPS Algorithm

```
# Name: Nolan Gregory
# Student ID: 30560013
# Extra Credit requirement performed successfully

.data
generated: .word 0, 0, 0
input: .word 0, 0, 0
inputString: .space 10

newline:.asciiz "\n"
comma:.asciiz ", "
bracketL:.asciiz "["
bracketR:.asciiz "]"
debug:.asciiz "DEBUG: "
prompt:.asciiz "Enter 3 digits (One line seperated by spaces): "
strikeString:.asciiz " strike(s) "
ballString:.asciiz " ball(s) "
outString:.asciiz "Out"
winPrompt:.asciiz "Good Job! Do you want to play again (y/n)?: "
winError:.asciiz "Please enter (Y/N)!\n"
addErrorDupeMessage:.asciiz "You cannot input duplicated numbers.\n"
addErrorRangeMessage:.asciiz "You must enter a value between [1, 9]!\n"

# Declare msciiz ain as a global function
.globl main

.text
# The label 'main' represents the starting point
main:
        la $a0, generated               #$a0 based address of array
        li $a1, 3                       #$a1 how many numbers are generated
        jal generate
        addi $t0, $zero, 0              #set loop variable to print array
        move $s0, $a0                   #set $s0 to be the base address for generated
        move $s1, $a1                   #set $s1 to be the const int 3
        addi $t8, $s1, -1               #set branch terminator at count - 1
        addi $s2, $zero, 1              #set while loop game logic to 1 (infinite loop)

formatPrint:
        li $v0, 4
        la $a0, debug                   #prints out the array for debug purposes
        syscall
        li $v0, 4
        la $a0, bracketL                #prints left bracket
        syscall
        j printArray

printArray:
        blt $t0, $t8, middlePrintArray
        move $a0, $s0                   #move base address back into $a0
        sll $t1, $t0, 2                 #bitshift to get offset
        add $t1, $t1, $a0               #get address of index
        lw $a0, 0($t1)                  #store number into arg1 for syscall
        li $v0, 1                       #print number
        syscall
        addi $t0, $t0, 1                #increment i by one
        j endPrintArray

middlePrintArray:
        move $a0, $s0                   #move base address back into $a0
        sll $t1, $t0, 2                 #bitshift to get offset
        add $t1, $t1, $a0               #get address of index
        lw $a0, 0($t1)                  #store number into arg1 for syscall
        li $v0, 1                       #print number
        syscall
        li $v0, 4
        la $a0, comma                   #print out a comma
        syscall
        addi $t0, $t0, 1                #increment i by one
        j printArray

endPrintArray:
        li $v0, 4
        la $a0, bracketR                #print out closing bracket
        syscall
        li $v0, 4
        la $a0, newline                 #print out newline
        syscall
        addi $t0, $zero, 0              #set $t0 to be zero

gameWhileLoop:
        bne $s2, 1, end                 #infinite while loop for game

getUserInput:
        la $s7, input
        sw $zero, 0($s7)                #Reinitialize input to be zeros
        sw $zero, 4($s7)                #Reinitialize input to be zeros
        sw $zero, 8($s7)                #Reinitialize input to be zeros
```

```
        li $v0, 4                       #Display prompt message
        la $a0, prompt
        syscall
        li $v0, 8
        la $a0, inputString             #read string into a0
        la $a1, 8                       #max buffer size
        move $t0, $a0                   #$t0 contains address of the sting
        syscall

loopThru:
        addi $t1, $zero, 0              #initialize $t1 in while loop
        addi $t2, $zero, 0              #initialize $t2 in while loop
        addi $t3, $zero, 0              #initialize $t3 in while loop
        addi $t4, $zero, 0              #initialize $t4 in while loop
        addi $t5, $zero, 0              #initialize $t5 in while loop

addToInput:
        bge $t1, 3, storedUserInput     #if i >= 3
        la $t0, inputString             #address of new string
        la $s0, input                   #address of inputarray
        sll $t2, $t1, 1                 #offset for string
        sll $t3, $t1, 2                 #offset for array
        add $t0, $t0, $t2               #position string
        add $s0, $s0, $t3               #position array
        lbu $t4, 0($t0)                 #load character
        andi $t4,$t4, 0x0F              #Mask to turn into integer
        la $a0, input                   #Set arg0 to input array for find()
        move $a1, $t4                   #Set arg1 to value for find()
        li $a2, 3                       #Set arg2 to count for find()
        jal find                        #Jump and link with find function
        bgt $t4, 9, addErrorRange
        blt $t4, 1, addErrorRange
        bne $v0, -1, addErrorDupe       #if return != -1, add error dupe
        sw $t4, 0($s0)                  #store into array
        addi $t1, $t1, 1                #increment i by 1
        j addToInput                    #go to top of loop

addErrorDupe:
        li $v0, 4
        la $a0, addErrorDupeMessage     #Display duplicate error message
        syscall
        j getUserInput

addErrorRange:
        li $v0, 4
        la $a0, addErrorRangeMessage    #Display range error message
        syscall
        j getUserInput

storedUserInput:
        la $a0, generated               #store generated in arg0
        la $a1, input                   #store input in arg1
        jal findBalls                   #call findBalls()
        move $s3, $v0                   #store ball count in $s3
        la $a0, generated               #store generated in arg0
        la $a1, input                   #store input in arg1
        jal findStrikes                 #call findStrikes()
        move $s4, $v0                   #store strike count in $s4

printStrikes:
        beq $s4, 0, printBalls          #if strikes == 0, print balls
        move $a0, $s4                   #set strikes to be printed
        li $v0, 1                       #print number
        syscall
        li $v0, 4                       #print string strikes
        la $a0, strikeString
        syscall

printBalls:
        beq $s3, 0, printOut            #if balls == 0, don't print
        move $a0, $s3                   #set balls to be printed
        li $v0, 1                       #print number
        syscall
        li $v0, 4                       #print string balls
        la $a0, ballString
        syscall

printOut:
        bne $s3, 0, gameLogic           #if strikes != 0 go to gamelogic
        bne $s4, 0, gameLogic           #if balls != 0 go to gamelogic
        li $v0, 4                       #print string out
        la $a0, outString               #else print out
        syscall

gameLogic:
        li $v0, 4                       #print newline
        la $a0, newline                 #print newline
        syscall
        beq $s4, 3, winGame             #if strikes == 3 go to wingame
        j gameWhileLoop

winGame:
        li $v0, 4
```

```
        la $a0, winPrompt               #prompt the user to enter if they want to play again
        syscall
        li $v0, 12                      #get a char from the user
        syscall
        move $t1, $v0                   #move char to $t1
        li $v0, 4                       #print newline
        la $a0, newline                 #print newline
        syscall                         #print newline
        beq $t1, 121, main              #check for valid input (case insensitive)
        beq $t1, 89, main               #check for valid input (case insensitive)
        beq $t1, 110, end               #check for valid input (case insensitive)
        beq $t1, 78, end                #check for valid input (case insensitive)
        li $v0, 4
        la $a0, winError                #print error message and reprompt the user for input
        syscall
        j winGame

#generate() function starts here
generate:
        addi $t2, $zero, 0              #sets loop variable ($t2) to zero
        addi $s0, $a1, 0                #set count to $s0
        move $s1, $a0                   #set base address to $s1
        addi $sp, $sp, -12              #open stack
        sw $ra, 0($sp)                  #store return address
        sw $a0, 4($sp)                  #store address of array
        sw $a1, 8($sp)                  #store count

whileAddedLessThanCount:
        beq $t2, $s0, endLoop           #if i == count, exit loop
        li $v0, 42                      #invoke syscall 42
        la $a1, 9                       #set random upper bound to 10
        syscall                         #$a0 is now set to the random value
        addi $a0, $a0, 1                #add 1 to randomNumber as to avoid 0
        move $a1, $a0                   #make random number the second argument
        move $a0, $s1                   #make address of array the first argument
        move $a2, $s0                   #make count the third argument
        jal find                        #find will return either -1 or the index of the number in the array
        addi $t0, $zero, 0              #set $t0 to store the return value of find
        addi $t0, $v0, 0                #set $t0 to store the return value of find
        beq $t0, -1, addValue           #if find returns -1, add the value to the array
        j whileAddedLessThanCount

addValue:
        sll $t3, $t2, 2                 #get offset of i
        add $t3, $t3, $a0               #get address of input[i]
        sw $a1, 0($t3)                  #store the value into arg1
        addi $t2, $t2, 1                #increment i by one
        j whileAddedLessThanCount

endLoop:
        lw $ra, 0($sp)                  #restore return address from stack
        lw $a0, 4($sp)                  #restore base address of array from stack
        lw $a1, 8($sp)                  #restore count from the stack
        addi $sp, $sp, 12               #close stack
        jr $ra                          #jump to main

#findBalls () function starts here
findBalls:
        addi $t0, $zero, 0              #set idx to zero;
        addi $t1, $zero, 0              #set num to zero
        addi $t2, $zero, 0              #set balls to zero
        addi $t3, $zero, 0              #set loop value to zero
        move $t4, $a0                   #set $t4 to address of generate
        move $t5, $a1                   #set $t5 to address of input
        addi $sp, $sp, -4               #open stack
        sw $ra, 0($sp)                  #store return address on stack
        move $a0, $a1                   #set $a0 to input for find function
        li $a2, 3                       #set a2 for count in function

findBallsLoop:
        beq $t3, 3, endFindBalls
        sll $t6, $t3, 2                 #offset for generated at [i]
        add $t6, $t6, $t4               #address of generated[i]
        lw $t6, 0($t6)                  #load generated[i] into $t6
        move $a1, $t6                   #set arg1 to generated[i]
        jal find
        move $t0, $v0                   #set index to be equal to return value from find
        beq $t0, -1, noBalls            #if index == 1, no balls found
        beq $t0, $t3, noBalls           #if index == i, stike not ball
        addi $t2, $t2, 1                #else, increment ballcount

noBalls:
        addi $t3, $t3, 1                #increment i by one
        j findBallsLoop

endFindBalls:
        lw $ra, 0($sp)                  #load return address
        addi $sp, $sp, 4                #close stack
        move $v0, $t2                   #set the return address to ballcount
        jr $ra                          #jump back to main

#findfindStrikes () function starts here
findStrikes:
```

```
        addi $t0, $zero, 0              #set idx to zero;
        addi $t1, $zero, 0              #set num to zero
        addi $t2, $zero, 0              #set strikecount to zero
        addi $t3, $zero, 0              #set loop value to zero
        move $t4, $a0                   #set $t4 to address of generate
        move $t5, $a1                   #set $t5 to address of input
        addi $sp, $sp, -4               #open stack
        sw $ra, 0($sp)                  #store return address on stack
        move $a0, $a1                   #set $a0 to input for find function
        li $a2, 3                       #set a2 for count in function

findStrikesLoop:
        beq $t3, 3, endFindStrikes
        sll $t6, $t3, 2                 #offset for generated at [i]
        add $t6, $t6, $t4               #address of generated[i]
        lw $t6, 0($t6)                  #load generated[i] into $t6
        move $a1, $t6                   #set arg1 to generated[i]
        jal find
        move $t0, $v0                   #set index to be equal to return value from find
        bne $t0, $t3, noStrikes         #if index != i, not a strike
        addi $t2, $t2, 1                #else, increment strikecount

noStrikes:
        addi $t3, $t3, 1
        j findStrikesLoop

endFindStrikes:
        lw $ra, 0($sp)                  #load return address
        addi $sp, $sp, 4                #close stack
        move $v0, $t2                   #setting return value
        jr $ra                          #jump back to main

find:
        li $t8, 0                       #set i to be zero
        li $v0, -1                      #set return value to zero

find_loop_start:
        bge $t8, $a2, find_loop_end     #check to see if i == count
        sll $t9, $t8, 2                 #get offset for array
        add $t9, $t9, $a0               #get address of element
        lw $t9, 0($t9)                  #load arr[i]
        beq $t9, $a1, find_loop_found   #check to see if arr[i] == num
        addi $t8, $t8, 1                #increment i by 1
        j find_loop_start               #go to top of loop

find_loop_found:
        addi $v0, $t8, 0                #if found, return index of array

find_loop_end:
        jr $ra                          #return to main

end:
        li $v0, 10                      # This system call terminates the program
        syscall
```

:: end program ::

MIPS Analysis

My MIPS algorithm begins with roughly a dozen references in the .data section. These include strings that are generate throughout the program, characters required to print the debug array properly, and an allocated buffer to store the input string. I begin the algorithm by calling the generate function which generates three random numbers in the range [1, 9] and stores in to an array called generated. Within this generate function I loop through and invoke the MIPS 42 system call which generates random numbers within a certain threshold [0, N) where N is the first argument to the system call. It is worth noting this as it required me to set the max threshold at 9 and add 1 to the return value as to force the value to be within the desired range. The function then calls the provided find function, which returns the index of an element if it is found within a provided array, or -1 if the index is not found within the array. This function gives me the ability to determine whether I can successfully add the randomly generated number to the array and increase the loop variable by one.

Upon returning from the generate function, my algorithm then goes through and prints the array in *formatted* order. This means that the print includes proper brackets (i.e. begins with a "[" and ends with a "]"), as well as proper comma delineation (i.e. 1,2,3 and **not** 1,2,3,). This is provided for debugging purposes as to easily test functionality of the program. Once the array has been printed, the user is then prompted to enter *3* values separated by a space (*this is very important as the input is read into a buffer and parsed through byte by byte*).

My getInput function has several key functions that I will attempt to explain succinctly. As mentioned, the input is read on one line (further information will be provided in the **extra credit** section), and parsed through looking at each individual character. I then *mask* the input value with 0x0F which turns the expression into the integer representation of the value the user input (similar behavior can be achieved by subtracting 48 from the ascii value). If the value the use entered is *not* within the range [1, 9], then the program prompts them to enter valid numbers and re-enter the data. Similarly, if the user inputs *duplicate* values, the program tells the user that duplicate values are not allowed. If neither of these error checks register, the program adds the *integer* representation to the array called input.

The program continues on by calling the findBalls and findStrikes functions. These two functions are relatively straightforward, and they simply invoke the provided find function to determine whether a value in the input array is in the generated array and generated[i] is not equal to input[i] (for findBalls), or that the value is in the generated array and generated[i] is equal to input[i]. I will not go over these functions in detail as they are rather simple.

Finally, the program goes through some game logic to determine the amount of balls, the amount of strikes, or if the user got an out. Likewise, the game logic determines if the user won the game by checking to see if the number of strikes is equal to the length of the generated array (in this case the const int 3). If this condition is true, the user is given the opportunity to restart the game or terminate the program. This input is case-insensitive and works by reading in a single char to the input buffer. If the user enters a char other than 'n' or 'y' (again, case sensitivity has been accounted for and can be ignored), then the program prompts the user to re-enter the char. If the user chooses to terminate the program, then the program invokes the system call to end the program, otherwise the array is regenerated and the user is prompted to enter more input.

Extra Credit

My program successfully implements the extra credit that was proposed in the handout. To accomplish this, I had to build some features that have not been discussed previously. One distinction is that I was required to use the .space directive in my .data section to allocate *buffer size* for my input string. Likewise, to read chars into this string, I had to use the MIPS syscall 8. The official MIPS documentation states that this syscall operates with the same functionality as the C fgets() function (that is, it ends with a newline character). An interesting feature I implemented was masking the input ascii value with 0x0F to get the integer representation. This was achieved by *and*-ing the value with 0x0F. This results in the value being *and*-ed with the binary representation 0000 1111, which results in producing the lower portion of the byte (thus providing us with the integer representation). I have used this in a similar fashion in C++ while bitshifting color values to extract their color channel representations, and I find it to be a fun way to work with logical operations. A similar representation can be achieved by subtracting the ascii value by 48 in this instance, but that is less exciting. In all, this still performs **all** vital error checking, and thus does not allow values outside the expected range or duplicate values.