# Baby Block

John Ingram

Thomas Bunney

Daniel Balding

Code:

// John Ingram

// Thomas Bunney

// Daniel Balding

#include <iostream>

#include <fstream>

#include <iomanip>

#include <string>

#define getNextBlock get\_block\_testthree() // Allows for changing this for test cases

// Functions

bool robot\_gtoreq\_slot(char robot, char in\_slot);

int compareBlocks(char robot, char in\_slot);

unsigned int move\_to(unsigned int target, unsigned int position);

unsigned int findSpot(char block, char array[], unsigned int position);

unsigned int nextEmpty(unsigned int position, char array[]); // moves the robot to the nearest empty slot

unsigned int placeBlock(char block, unsigned int position, char array[]); // places block in 'array' at 'position'

char get\_block(void);

void print\_slots(char slots[]);

unsigned int put\_block(char block, unsigned int position, char array[]);

unsigned int remove\_block(unsigned int position, char array[]);

unsigned int shift\_right(unsigned int position);

unsigned int shift\_left(unsigned int position);

bool robot\_ltoreq\_slot(char robot, char in\_slot);

char switch\_blocks(char robot, unsigned int position, char array[]);

bool test\_empty(unsigned int position, char array[]);

char get\_block\_testcase(unsigned int testcase, unsigned int index);

char get\_block\_testone(void);

char get\_block\_testtwo(void);

char get\_block\_testthree(void);

char get\_block\_testfour(void);

char get\_block\_testfive(void);

using namespace std;

unsigned int switchCount = 0; //TODO: Implement counting the number of switches made

int main(void)

{

    unsigned int position = 10, placed = 0;

    char array[20] = {};

    char robot;

    //initial setup

    for (int i = 0; i < 20; i++) array[i] = NULL;

    // place first block

    robot = getNextBlock;

    put\_block(robot, position, array);

    placed++;

    print\_slots(array);

    cout<< "Number of blocks switched: "<<switchCount<<endl;

    while (placed < 20)

    {

        robot = getNextBlock;

        position = findSpot(robot, array, position);

        position = placeBlock(robot, position, array);

        placed++;

        print\_slots(array);

        cout<< "Number of blocks switched: "<<switchCount<<endl;

    }

    system("pause");

    return 0;

}

// Functions

bool robot\_gtoreq\_slot(char robot, char in\_slot)

{

    bool debug = false;

    if (debug)

        cout << endl << "Comparing robot block " << robot << " with block in slot " << in\_slot << endl;

    if (robot >= in\_slot)

    {

        if (debug)

            cout << "Returning true. Robot block GREATER than or EQUAL to block in slot. " << endl;

        return true;

    }

    else

    {

        if (debug)

            cout << "Returning false. Robot block LESS than block in slot. " << endl;

        return false;

    }

}

// returns -1 when 'robot' is < 'in\_slot'

// returns 0 when 'robot' is == 'in\_slot'

// returns 1 when 'robot' is > 'in\_slot'

// returns 9999 when 'in\_slot' is empty

int compareBlocks(char robot, char in\_slot)

{

    if (in\_slot == NULL) return 9999;

    if (robot\_ltoreq\_slot(robot, in\_slot) == true)

    {

        if (robot\_gtoreq\_slot(robot, in\_slot) == true) return 0;

        else

        {

            return -1;

        }

    }

    else return 1;

}

unsigned int findSpot(char block, char array[], unsigned int position)

{

    position = move\_to(0, position);

    int i = 0;

    while (i < 19)

    {

        if (test\_empty(position, array))

        {

            position = shift\_right(position);

        }

        if (compareBlocks(block, array[position]) == 0) return position;

        if (compareBlocks(block, array[position]) == -1)

        {

            if(position == 0) return 0;

            return position - 1;

        }

        position = shift\_right(position);

        i++;

    }

    return position;

}

unsigned int move\_to(unsigned int target, unsigned int position) // input must be =<19 and >=0

{

    while (true)

    {

        if (target < position) position = shift\_left(position);

        else if (target > position) position = shift\_right(position);

        else return position;

    }

}

unsigned int nextEmpty(unsigned int position, char array[])// moves the robot to the nearest empty slot

{

    unsigned int nextLeft, nextRight, oldPosition = position;

    //Look to the left

    while (!test\_empty(position, array) && position != 0)

    {

        position = shift\_left(position);

    }

    if (test\_empty(position, array)) nextLeft = position;

    else nextLeft = 9999;

    //reset position

    position = move\_to(oldPosition, position);

    //Look to the right

    while (!test\_empty(position, array) && position != 19)

    {

        position = shift\_right(position);

    }

    if (test\_empty(position, array)) nextRight = position;

    else nextRight = 9999;

    // Calculate the smaller distance, and move there

    if (nextLeft == 9999)

    {

        position = move\_to(nextRight, position);

        return position;

    }

    if (nextRight == 9999)

    {

        position = move\_to(nextLeft, position);

        return position;

    }

    unsigned int deltaLeft = oldPosition - nextLeft;

    unsigned int deltaRight = nextRight - oldPosition;

    if (deltaLeft < deltaRight) position = move\_to(nextLeft, position);

    if (deltaLeft > deltaRight) position = move\_to(nextRight, position);

    return position;

}

unsigned int placeBlock(char block, unsigned int position, char array[]) // places block in 'array' at 'position'

{

    int direction = 0;

    unsigned int empty, oldPosition = position;

    if (test\_empty(position, array))

    {

        put\_block(block, position, array);

        return position;

    }

    empty = position = nextEmpty(position, array);

    position = move\_to(oldPosition, position);

    if (empty < position)

    {

        while (position != empty)

        {

            block = switch\_blocks(block, position, array);

            switchCount++;

            position = shift\_left(position);

        }

        position = put\_block(block, position, array);

        return position;

    }

    if (empty > position)

    {

        while (position != empty)

        {

            block = switch\_blocks(block, position, array);

            switchCount++;

            position = shift\_right(position);

        }

        position = put\_block(block, position, array);

        return position;

    }

}

// ------------------------------------------------------------------------ //

//                                                                          //

//                                                                          //

//              Instructor provided Functions for robot operations          //

//                                                                          //

//                                                                          //

// ------------------------------------------------------------------------ //

//

// Function get\_block

// Reads in a single character value from the keyboard

// This is the input from the chute

// Returns: a single character in uppercase

//

// Example function call:   block = get\_block();

char get\_block(void)

{

    char block;

    cout << "Enter one block: ";

    cin >> block;

    return toupper(block);

}

//

// Function print\_slots

// Prints the contents of the slots array in a well formatted form.

// Input: Array of slots

// Returns: Nothing (void)

//

// Example function call: print\_slots(slot\_array);

void print\_slots(char slots[])

{

    unsigned int j = 1;

    for (j = 1; j <= 20; j++)

    {

        cout << setw(3) << j;

    }

    cout << endl;

    for (j = 0; j < 20; j++)

    {

        cout << setw(3) << slots[j];

    }

    cout << endl;

}

// Function put\_block

// This function stores a character into the character array representing the slots

//

// Inputs:

// block - type char - The character to be inserted into a slot

// position - type unsigned int - index of the slot where the block will go

// array - type char - array of slots containing the blocks

//

// Returns:

// position - type unsigned int - the index of the slot where the block was placed

//

// Example function call:   put\_block(block, position, slots);

unsigned int put\_block(char block, unsigned int position, char array[])

{

    bool debug = false;

    array[position] = block;

    if (debug)

        cout << "Block " << block << " inserted into slot " << position << endl;

    return position;

}

// Function remove\_block

// This function removes a block from the slot array

// The slot where the block is removed is then set to a space

//

// Inputs:

// position - type unsigned int - index of the slot where block is located

// array - type char - array of slots containing the blocks

//

// Returns:

// block - type char - the block removed from the slot

//

// Example function call:   remove\_block(position, slots);

unsigned int remove\_block(unsigned int position, char array[])

{

    bool debug = false;

    char block = ' ';

    block = array[position];

    array[position] = ' ';  // Reset slot to blank after block removed

    if (debug)

        cout << "Block " << block << " removed from slot " << position + 1 << endl;

    return block;

}

// Function shift\_right

// This function increments the index simulating a movement of the robot

// to the next higher slot (index) of the array

//

// Inputs:

// position - type unsigned int - current slot position

//

// Returns:

// position - type unsigned int - The updated position which is input position + 1

//

// Example function call:  position = shift\_right(position)

//

unsigned int shift\_right(unsigned int position)

{

    bool debug = false;

    if (position < 19)

    {

        position++;

        if (debug)

            cout << "Position right shifted to " << position << endl;

    }

    return position;

}

// Function shift\_left

// This function decrements the index simulating a movement of the robot

// to the next lower slot (index) of the array

//

// Inputs:

// position - type unsigned int - current slot position

//

// Returns:

// position - type unsigned int - The updated position which is input position - 1

//

// Example function call: position = shift\_left(position)

//

unsigned int shift\_left(unsigned int position)

{

    bool debug = false;

    if (position > 0) position--;

    {

        if (debug)

            cout << "Position left shifted to " << position << endl;

    }

    return position;

}

// Function robot\_ltoreq\_slot

// This function compares the value of the block held by the robot

// with the value of the block in a slot

//

// Inputs:

// robot - type char - value of block held by robot

// in\_slot - type char - value of block in the slot

//

// Returns:

// true or false

// TRUE if block held by robot is LESS than or equal to the block in slot

// FALSE if block held by robot is GREATER than block in slot

//

// Example function call: if ( compare\_blocks(robot\_block, slot\_block) )

//

bool robot\_ltoreq\_slot(char robot, char in\_slot)

{

    bool debug = false;

    if (debug)

        cout << endl << "Comparing robot block " << robot << " with block in slot " << in\_slot << endl;

    if (robot <= in\_slot)

    {

        if (debug)

            cout << "Returning true. Robot block LESS than or EQUAL to block in slot. " << endl;

        return true;

    }

    else

    {

        if (debug)

            cout << "Returning false. Robot block GREATER than block in slot. " << endl;

        return false;

    }

}

// Function switch\_blocks

// This function switches the block held by the robot with a block in a slot.

// After the switch the robot is holding the block removed from the slot.

//

// Inputs:

// robot - type char - The block to be inserted into a slot

// position - type unsigned int - index of the slot where the block will go

// array - type char - array of slots containing the blocks

//

// Returns:

// robot - type char. The value of the block removed from the slot.

//

// Example function call: block = switch\_blocks(block,  position, array);

//

char switch\_blocks(char robot, unsigned int position, char array[])

{

    char temp\_hold;

    bool debug = false;

    if (debug)

        cout << "Switching blocks " << robot << " with " << array[position] << endl;

    temp\_hold = robot;

    robot = array[position];

    array[position] = temp\_hold;

    return robot;

}

// Function test\_empty

// This function tests the array to determine if a slot is empty (NULL)

// or if the slot contains a blank. The slot array must be intialized to

// all NULL or all blanks (spaces) before any blocks are added.

//

// Inputs:

// position - type unsigned int - index of slot to be tested

//

// Returns:

// true or false as value o function

// TRUE if slot is empty

// FALSE if the slot contains a block

//

// Example function call: if ( test\_empty(index, array) )

//

bool test\_empty(unsigned int position, char array[])

{

    char blank = ' '; // Blank space

    bool debug = false;

    if (array[position] == NULL || array[position] == blank)

    {

        if (debug)

            cout << "Slot " << position << " empty. " << endl;

        return true;

    }

    else

    {

        if (debug)

            cout << "Slot " << position << " contains a block " << endl;

        return false;

    }

}

// ------------------------------------------------------------------------ //

//                                                                          //

//                                                                          //

//              Functions for robot test cases                              //

//                                                                          //

//                                                                          //

// ------------------------------------------------------------------------ //

//

// Function get\_block\_testcase

// Returns blocks for a given test case one at a time

// This function can be used for all five test cases.

//

// Input: Unsigned int case designating test case 1-5.

// Input: Index of letter desired 0 - 19.

// Returns: a single uppercase character

//

// Example function call:   block = get\_block\_testcase(1, 0);

// Returns first letter of test case 1.

char get\_block\_testcase(unsigned int testcase, unsigned int index)

{

    string testcases[5] = { "AXFIUTRPQVWSEYJINYTB",

        "ABFGHIJKMOPRSTUVWXYZ",

        "ZYXWVUTSRPOKJIIHGFBA",

        "AAAAAYYYYYQQQQQXXXXX",

        "XXXAAAZZZAAYYVVVVQQQ" };

    return testcases[testcase - 1].at(index);

}

//

// Function get\_block\_testone

// Returns blocks for test case one one at a time

// Returns: a single character

//

// Example function call:   block = get\_block\_testone();

char get\_block\_testone(void)

{

    static unsigned int index = 0;

    char test\_case\_one[21] = "AXFIUTRPQVWSEYJINYTB";

    return test\_case\_one[index++];

}

//

// Function get\_block\_testtwo

// Returns blocks for test case two one at a time

// Returns: a single character

//

// Example function call:   block = get\_block\_testtwo();

char get\_block\_testtwo(void)

{

    static unsigned int index = 0;

    char test\_case\_two[21] = "ABFGHIJKMOPRSTUVWXYZ";

    return test\_case\_two[index++];

}

//

// Function get\_block\_testthree

// Returns blocks for test case three one at a time

// Returns: a single character

//

// Example function call:   block = get\_block\_testthree();

char get\_block\_testthree(void)

{

    static unsigned int index = 0;

    char test\_case\_three[21] = "ZYXWVUTSRPOKJIIHGFBA";

    return test\_case\_three[index++];

}

//

// Function get\_block\_testfour

// Returns blocks for test case four one at a time

// Returns: a single character

//

// Example function call:   block = get\_block\_testfour();

char get\_block\_testfour(void)

{

    static unsigned int index = 0;

    char test\_case\_four[21] = "AAAAAYYYYYQQQQQXXXXX";

    return test\_case\_four[index++];

}

//

// Function get\_block\_testfive

// Returns blocks for test case five one at a time

// Returns: a single character

//

// Example function call:   block = get\_block\_testfive();

char get\_block\_testfive(void)

{

    static unsigned int index = 0;

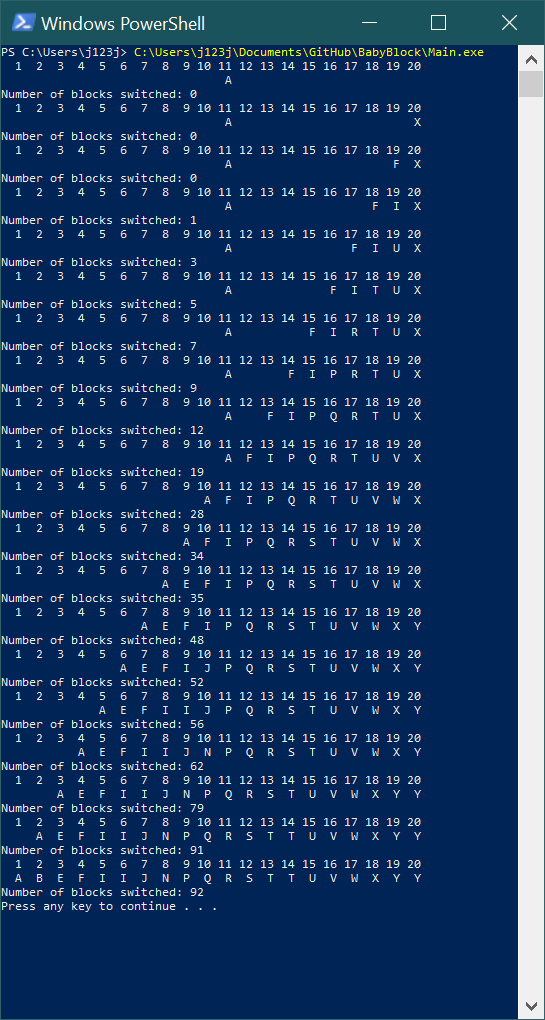
    char test\_case\_five[21] = "XXXAAAZZZAAYYVVVVQQQ";

    return test\_case\_five[index++];

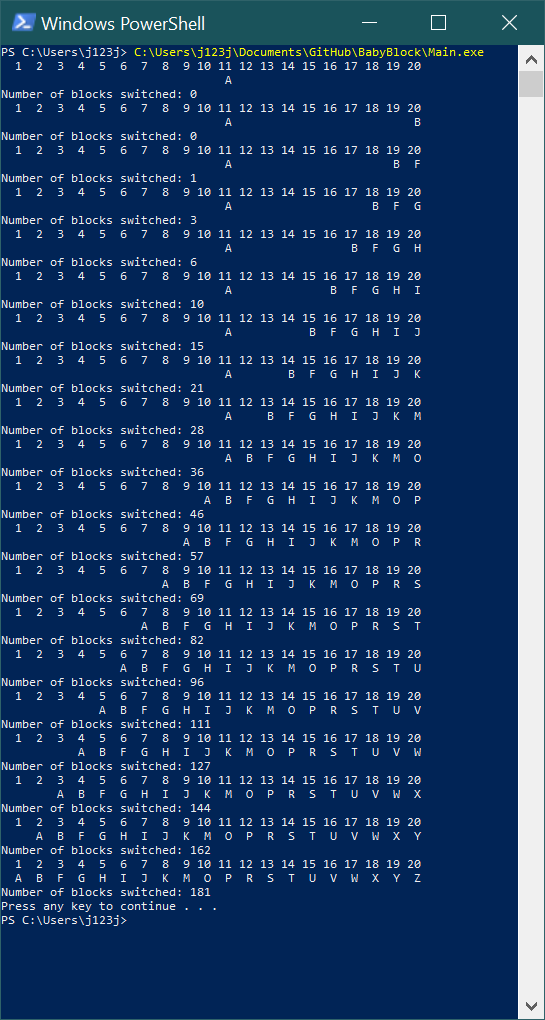
}

**Test Cases:**

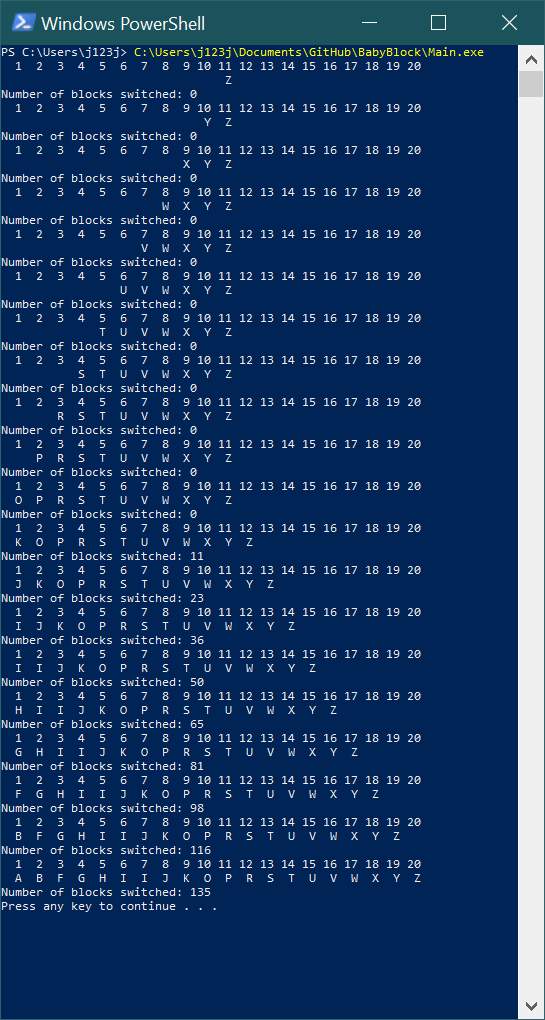
Test case 1:



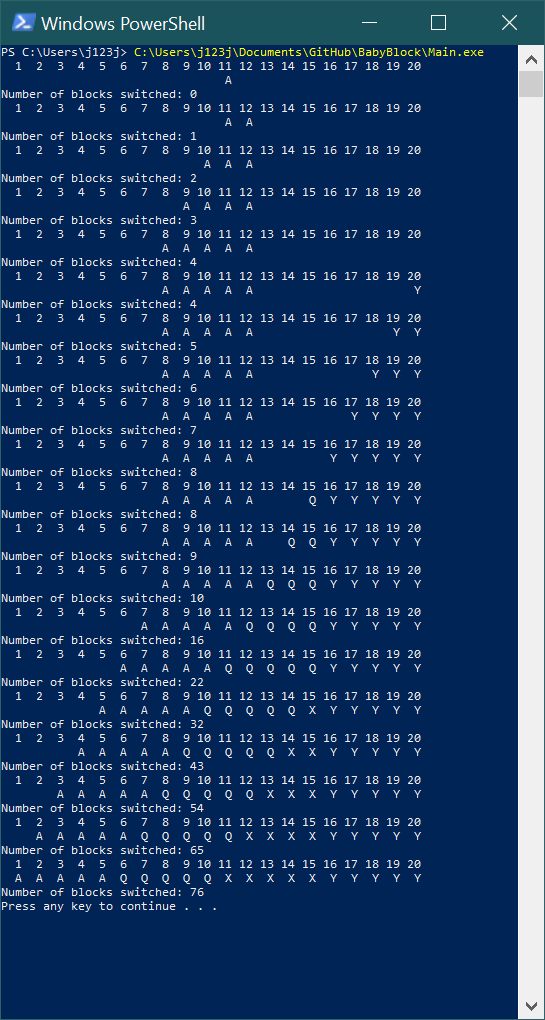
Test Case 2:



Test Case 3:



Test Case 4:



Test Case 5:

