WVU CS350 Spring-2010
Assignment 1

Source Code Printout

Paul Prince

Table of Contents

Question 1

```c
/*
 * Program to find Pythagorean Triples
 * ===================================
 *      By:     Paul A. Prince
 *      For:    WVU CS 350 - Donald Adjeroh
 *      Date:   2010-02-10 (created)
 *
 * Assignment 1, Question 1:
 * -------------------------
 *
 * Pythagorean triple is: set of 3 integers (a, b, c) where a^2 + b^2 = c^2
 *
 * Finds all such triples where none of the 3 integers are <1 or >500.
 *
 * Treats (3, 4, 5) and (4, 3, 5) as distinct triples.
 */

#include <stdio.h>
#include <math.h>


int main(void) {

        int a, b;               /* the two sides (legs) of the right triangle, */
        int c;                  /* the hypotenuse. */

        int num_found = 0;      /* keep track of how many pythagorean triples we find. */

        for (a=1; a<=500; a++) {
                for (b=1; b<=500; b++) {
                        for (c=1; c<=500; c++) {
                                /*    if a^2  +     b^2  ==      c^2,        */
                                if ( pow(a,2) + pow(b,2) == pow(c,2) ) {
                                /*    then we have a pythagorean triple: */
                                        num_found++;
                                        printf("result %3d : a= %3d, b= %3d, c= %3d\n", num_found, a, b, c);
                                }
                        }
                }
        }

        return 0; /* exit successfully */
}
```

Question 2

```
/*
 * Program to find Perfect Numbers
 * ===============================
 *      By:     Paul A. Prince
 *      For:    WVU CS 350 - Donald Adjeroh
 *      Date:   2010-02-10 (created)
 *
 * Assignment 1, Question 2:
 * -------------------------
 *
 * Perfect number is: integer whose factors including 1 (but not the integer itself) sum to the integer.
 * Examples:  (http://en.wikipedia.org/wiki/List_of_perfect_numbers)
 *     6 = 1 + 2 + 3
 *    28 = 1 + 2 + 4 + 7 + 14
 *
 * Finds all such numbers between 1 and 10000.
 */

#include <stdio.h>


int main(void) {
        int n;
        for (n=1; n<=10000; n++) {
                int sum = 0;
                int i;
                for ( i=1; i<n; i++ ) {
                        if (n%i==0) sum+=i;
                }
                if (n==sum) printf("result: %4d\n", n);
        }

        printf("Stopping after %d attempts.\n", (n - 1));

        return 0; /* exit successfully */
}
```

Question 3

```c
/*
 * Program to find Greatest Common Divisor
 * ===========================================
 *      By:     Paul A. Prince
 *      For:    WVU CS 350 - Donald Adjeroh
 *      Date:   2010-02-10 (created)
 *
 * Assignment 1, Question :3
 * -------------------------
 *
 */

#include <stdio.h>
#include <stdlib.h>


#include "prompt.c"


int gcd_recursive(int x, int y);
int gcd_iterative(int x, int y);


int main(void) {

        int x;
        int y;

        printf("\nFind Greatest Common Divisor...\n");
        printf(  "================================\n\n");

        x = int_prompt("Enter 1st value (X)");
        y = int_prompt("Enter 2nd value (Y)");

        printf("\n  Values of ( X , Y ) are ( %d , %d ).\n\n", x, y);
        printf("     ** Recursive algorithm result: %8d **\n", gcd_recursive(x, y));
        printf("     ** Iterative algorithm result: %8d **\n", gcd_iterative(x, y));

        printf("\n");
        return 0; /* exit successfully */
}
```

```c
int gcd_iterative(int x, int y) {
        /* Euclidean algorithm */
        /* http://en.literateprograms.org/Euclidean_algorithm_(C) */
        if (y > x) { int t = x; x = y; y = t; }
        while (y != 0) {
                int m = x % y;
                x = y;
                y = m;
        }
        return x;
}


int gcd_recursive(int x, int y) {
        if ( y == 0 ) {
                return x;
        }
        return gcd_recursive( y, x % y );

}
```

Question 4

```c
/*
 * Program to find Knight's Tours of a Chessboard
 * ==============================================
 *      By:     Paul A. Prince
 *      For:    WVU CS 350 - Donald Adjeroh
 *      Date:   2010-02-10 (created)
 *
 * Assignment 1, Question :4
 * -------------------------
 *
 */

#include <stdio.h>
#include <stdlib.h>


#include "prompt.c"


void init_chessboard(int chessboard[8][8]);
void build_accessibility(int chessboard[8][8], int accessibility[8][8]);
void print_chessboard(int chessboard[8][8]);
int is_valid_move( int chessboard[8][8], int row, int col, int move );
int move_row( int row, int move );
int move_col( int col, int move );
```

```c
int main(void) {

        /* keep track of Knight's current position */
        int row, col;

        /* 8x8 array to represent the chessboard, */
        int chessboard[8][8];

        /* and an 8x8 array for the accessibility heuristic employed. */
        int accessibility[8][8];

        /* initialize our chessboard (empty board by default) */
        init_chessboard( chessboard );

        /* build the initial accessibility table for an empty chessboard */
        build_accessibility( chessboard, accessibility );


        /* Display some stuff to the user... */

        printf("\n        ==================\n");
        printf(  "        = KNIGHT'S TOUR =\n");
        printf(  "        ==================\n\n\n");

        printf(" An Empty Chessboard:\n");
        printf(" ================================\n");

        print_chessboard( chessboard );

        printf("\n\n");

        printf(" Accessibility for Empty Board:\n");
        printf(" ================================\n");

        print_chessboard( accessibility );

        printf("\n\n");

        row = int_prompt("Enter Knight's starting row   ");
        col = int_prompt("Enter Knight's starting column");

        /* catch invalid user input. */
        if ( row < 0 || row > 7 || col < 0 || col > 7 ) {
                printf("\n\n!!! Error:  Invalid Knight's starting position specified.  Exiting.\n");
                printf("            Rows and columns are numbered from 0 to 7.\n");
                exit(1);
        }
```

```
/* place the Knight on his starting position */
chessboard[row][col] = -1;

/* rebuild the accessibility table, now that the Knight has been placed on the board
 * (the board is no longer empty.) */
build_accessibility( chessboard, accessibility );



int move_counter = 0;
while(1) {

        printf(" ------------------------------\n\n");

        int move;
        int best_move                 = -1; /* magic value! */
        int best_move_accessibility   = 10; /* magic value! */
        for(move=0; move<8; move++) {
                if ( is_valid_move(chessboard, row, col, move) ) {
                        if ( accessibility[ move_row(row, move) ][ move_col(col, move) ] < best_move_accessibility ) {
                                best_move = move;
                                best_move_accessibility = accessibility[ move_row(row, move) ][ move_col(col, move) ];
                        }
                }
        }

        if ( best_move == -1 ) {
                if ( move_counter == 63 ) {
                        printf("\n   Hurrah, A tour was found!\n\n");
                        exit(0);
                } else {
                        printf("failure: no more moves possible after %d moves.\n", move_counter);
                        exit(1);
                }
        }

        move_counter++;
        chessboard[row][col] = move_counter;
        row = move_row(row, best_move);
        col = move_col(col, best_move);
        chessboard[row][col] = -1;
        build_accessibility( chessboard, accessibility );

        printf(" Chessboard: (move %2d)\n", move_counter);
        printf(" ==============================\n");
        print_chessboard( chessboard );
        printf("\n\n");
```

```
                printf(" Table of Accessibility Numbers:\n");
                printf(" ================================\n");
                print_chessboard( accessibility );
                printf("\n\n");
        }

        return 0; /* exit successfully */
}


void init_chessboard(int chessboard[8][8]) {
        int i, j;
        for (i=0; i<8; i++) {
                for (j=0; j<8; j++) {
                        chessboard[i][j]=0;
                }
        }
}


void build_accessibility(int chessboard[8][8], int accessibility[8][8]) {
        int i, j, k;
        for (i=0; i<8; i++) {
                for (j=0; j<8; j++) {
                        int squares_accessibility = 0;
                        for (k=0; k<8; k++) {
                                if ( is_valid_move(chessboard, i, j, k) ) {
                                        squares_accessibility++;
                                }
                        }
                        accessibility[i][j] = squares_accessibility;
                }
        }
}
```

```
int move_row( int row, int move ) {
        int horizontal[8];
        horizontal[ 0 ] =  2; horizontal[ 1 ] =  1; horizontal[ 2 ] = -1; horizontal[ 3 ] = -2;
        horizontal[ 4 ] = -2; horizontal[ 5 ] = -1; horizontal[ 6 ] =  1; horizontal[ 7 ] =  2;
        return row + horizontal[ move ];
}


int move_col( int col, int move ) {
        int vertical[8];
        vertical[ 0 ] = -1; vertical[ 1 ] = -2; vertical[ 2 ] = -2; vertical[ 3 ] = -1;
        vertical[ 4 ] =  1; vertical[ 5 ] =  2; vertical[ 6 ] =  2; vertical[ 7 ] =  1;
        return col + vertical[ move ];
}


int is_valid_move( int chessboard[8][8], int row, int col, int move ) {
        row = move_row( row, move );
        col = move_col( col, move );

        if ( row > 7 || row < 0 || col < 0 || col > 7 ) {
                return 0; // false; not a valid move
        } else if ( chessboard[row][col] != 0 ) {
                return 0; // false; not a valid move
                          // can only move to an unoccupied square.
        } else {
                return 1; // true; is a valid move
        }
}
```

```
void print_chessboard(int chessboard[8][8]) {
        int i, j;
        for (i=0; i<8; i++) {
                printf("+---+---+---+---+---+---+---+---+\n");
                for (j=0; j<8; j++) {
                        switch (chessboard[i][j]) {
                                case -1: /* Knight is on this square */
                                        printf("| K ");
                                        break;
                                case  0: /* define 0 as 'empty square' value */
                                        printf("|   ");
                                        break;
                                case -2: /* special case: print row/col numbers */
                                        printf("|%d,%d", i, j);
                                        break;
                                default: /* normally, just print the value in the square */
                                        printf("|%2d ", chessboard[i][j]);
                                        break;
                        }
                }
                printf("|\n");
        }
        printf("+---+---+---+---+---+---+---+---+\n");
}
```

```c
/*
 * Simple routines to prompt the user for input (using scanf)
 * ==========================================================
 *      By:     Paul A. Prince
 *      For:    WVU CS 350 - Donald Adjeroh
 *      Date:   2010-02-10 (created)
 *
 * This file is intended as a sort of library.
 */

#include <stdio.h>
#include <stdlib.h>


int int_prompt(char* prompt_string) {
        /* scanf() can be tricky to make behave properly.
         * reference: http://bytes.com/topic/c/answers/212167-help-infinite-loops-scanf */

        int prompt_answer;
        int scanf_return;

        while(1) {
                printf("        ");
                printf(prompt_string);
                printf(" : ");

                fflush(stdout);

                scanf_return = scanf("%d%*[^\n]", &prompt_answer);

                if ( scanf_return == 1 ) {
                        return(prompt_answer);
                } else {
                        clearerr(stdin);
                        scanf("%*[^\n]");
                }
        }
}
```