**Spring 2020: Chp5 & 6 Deitel: Functions and Arrays : pr66**

**Deitel/Subbarao: Modified 02 17 2020: Copy right extended thru 2020**

// : Chp5 & 6 Deitel: Functions and Arrays: Modified AND Extended THRU Spring 2020

// Subbarao V Wunnava: Courtesy Deitel: Setting up and Math operations on Arrays

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

//declare printArray function and addArray functions and subtractArrayColumn function

void printArray ( int x[3][3]); //function proto type to print array

void addArray (int y[3][3], int z[3][3]); // function proto type to add arrays

void subtractArrayColumn (int p[3][3], int q[3][3]); //function proto type to subtract arrays by columns

int main(void)

{

//initialize A and B arrays

int A [3][3] = { {2, 4, 6 }, {1, 3, 5}, {7, 8, 9} };

int B [3][3] = { {2, 7}, {4, 3, 1}, {9} };

int C [3][3] = { {1}, {1,2}, {2,4} };

//printing arrays A, B, and C by rows

printf ("\n\a Subbarao/Deitel Chp 5&6: Functions: Multi Dimensional Arrays: \n");

printf ("\n\a pgs 195-250 F14 2880 09 11 2014 Modified 09 18 2014 : \n");

puts("\n Array A by row :\n ");

printArray (A);

puts("\n Array B by row :\n ");

printArray (B);

puts("\n Array C by row :\n ");

printArray (C);

// adding and subtracting arrays

puts("\n add A and B arrays :\n ");

addArray (A,B);

puts("\n add B and C arrays :\n ");

addArray (B,C);

puts("\n subtract B and C arrays by columns :\n ");

subtractArrayColumn (B,C);

} // end main

//start of printArray to output the Arrays

void printArray ( int x[3][3])

{

size\_t r; // row counter

size\_t c; // column counter

// loop through rows

for ( r = 0; r <=2; ++r) //outer loop for rows

{

for ( c= 0; c <=2; ++c) // inner loop for columns

{

printf (" \t %d", x[r][c]);

}

printf ("\n\a" );

}

} //end printArray function

// start of the addArray function to add specified arrays

void addArray ( int y[3][3], int z[3][3])

{

size\_t l; // row counter

size\_t m; // column counter

// loop through rows

for ( l = 0; l <=2; ++l) //outer loop for rows

{

for ( m= 0; m <=2; ++m) // inner loop for columns

{

printf (" \t %d", (y[l][m]+ z[l][m]));

}

printf ("\n\a" );

}

} //end addArray function

// start of the subtractArray function to subtract specified arrays by columns

void subtractArrayColumn ( int p[3][3], int q[3][3])

{

size\_t k; // row counter

size\_t j; // column counter

// loop through rows and then columns

for ( k = 0; k <=2; ++k) //outer loop for rows

{

for ( j= 0; j <=2; ++j) // inner loop for columns

{

printf (" \t %d", (p[k][j]- q[k][j]));

}

printf ("\n\a" );

}

// loop through columns and then rows

printf ("\n loop through columns and then rows \n" );

for ( k = 0; k <=2; ++k) //outer loop for rows

{

for ( j= 0; j <=2; ++j) // inner loop for columns

{

printf (" \t %d", (p[j][k]- q[j][k]));

}

printf ("\n\a" );

}

// loop through columns and then rows in criss cross way

printf ("\n loop through columns and then rows in criss cross way \n" );

for ( k = 0; k <=2; ++k) //outer loop for rows

{

for ( j= 0; j <=2; ++j) // inner loop for columns

{

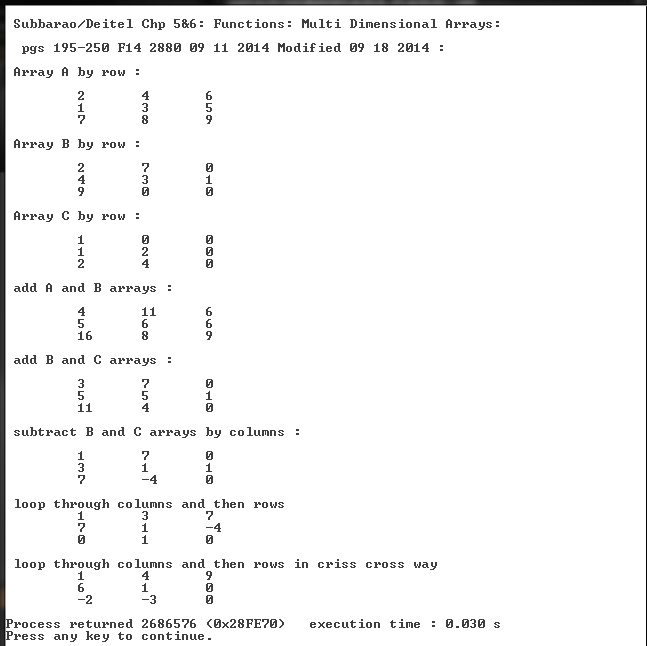
printf (" \t %d", (p[j][k]- q[k][j]));

}

printf ("\n\a" );

}

} //end subtractArrayColumn function



**6.19 *(Dice Rolling)* Write a program that simulates the rolling of two dice. The program should**

**use rand twice to roll the first die and second die, respectively. The sum of the two values should**

**then be calculated. [*Note:* Because each die can show an integer value from 1 to 6, then the sum of**

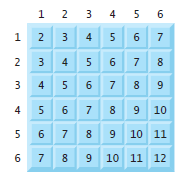
**the two values will vary from 2 to 12, with 7 being the most frequent sum and 2 and 12 the least**

**frequent sums.] Figure 6.24 shows the 36 possible combinations of the two dice. Your program**

**should roll the two dice 36,000 times. Use a single-subscripted array to tally the numbers of times**

**each possible sum appears. Print the results in a tabular format. Also, determine if the totals are**

**reasonable; i.e., there are six ways to roll a 7, so approximately one-sixth of all the rolls should be 7.**

****

// Exercise 6.19 Solution: Deitel/Subbarao Modified Spring 2020: 02 17 2020

// EEL 2880 Original Patent 09 18 2016/Modified 02 17 2020 Random Dice rolling

// and array computations

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

int main( void )

{

unsigned int i; // loop counter

unsigned int j; // loop counter

unsigned int x; // first die

unsigned int y; // second die

unsigned int sum[ 13 ] = { 0 }; // count occurrences of each combination

// array expected contains counts for the expected

// number of times each sum occurs in 36 rolls of the dice

unsigned int expected[ 13 ] = {0, 0, 1, 2, 3, 4, 5, 6, 5, 4, 3, 2, 1};

printf("\n\n\a\a EEL 2880 Fall 2013 09 18 2013: Random Dice rolling \n\n ");

printf(" and array computations: Problem 6.19 Deitel/Subbarao \n\n");

srand( time( NULL ) ); // seed random number generator

// roll dice 36,000 times using -for loop 1-

for ( i = 1; i <= 36000; ++i )

{

x = 1 + rand() % 6;

y = 1 + rand() % 6;

++sum[ x + y ];

} // end for-1

printf ("\n\t final i value: %u \n\n ", i);

printf( "%10s%10s%10s%10s\n", "Sum", "Total", "Expected", "Actual" );

// display results of rolling dice using for loop 2

for ( j = 2; j <= 12; ++j )

{

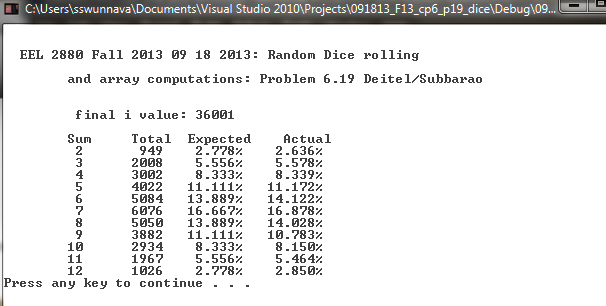
printf( "%10u%10u%9.3f%%%9.3f%%\n", j, sum[ j ],

100.0 \* expected[ j ] / 36, 100.0 \* sum[ j ] / 36000 );

} // end for-2

system("PAUSE");

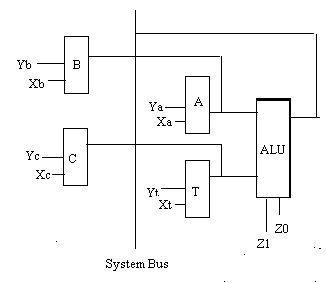
} // end main



**EEL 2880: Introduction to Machine Code and Micro Architecture: Spring 2020:**

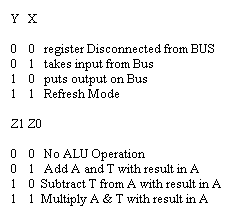
**Ref: Chapters 6 and 7 Deitel: : Arrays&Pointers Fall 2016 Updated 02 17 2020**

Inside a microprocessor/computer system, nearly 80-85 % of the time, data transfers take place on the system bus. The machine code operation controls the data flow and any arithmetic and logical operations.

**Courtesy: INTEL CORPORATION**

A: Accumulator T : Temporary Register; ALU: Arithmetic Logic Unit B & C are registers

Y and X are data transfer control signals and Z1 Z0 are ALU control signals for machine code



Control word is: Z1 Z0 Ya Xa Yt Xt Yb Xb Yc Xc Hex Code

A 🡨 B 0 0 0 1 0 0 1 0 0 0 048

T 🡨 C 0 0 0 0 0 1 0 0 1 0 012

A 🡨 A+T 0 1 0 1 0 0 0 0 0 0 140

B 🡨 A 0 0 1 0 0 0 0 1 0 0 084

The above operation is B = B+C in higher level languages