Talen Phillips

EE107-01

Homework 6

Due: 01APR2013

1) The Fibonacci numbers are 0,1,1,2,3,5,8,13,… where each number is the sum of the two preceding numbers. Write a program that declares an array named *fib\_numbers* of length 40 and fills the array with the first 40 Fibonacci numbers using a loop. Then prints the numbers to the screen.

/\* Talen Phillips

\* EE107-01

\* 01APR2013

\* "Fibonacci"

\*/

#include <stdio.h>

int main (void)

{

int i, fib\_numbers[40]; //initialize i and a 1D, 40 variable array.

/\* this loop calculates the fibonacci numbers 1-40 and stores them

\* in an array. If it attempts to continue, it will overflow the

\* int variable at the 48th iteration.\*/

for (i = 0;i < 40;i++) {

if ((i-2) < 0) fib\_numbers[i] = i; //starts the sequence {0,1}

else fib\_numbers[i] = fib\_numbers[i-1] + fib\_numbers[i-2];

}

/\* this loop displays the array in 5 columns.\*/

for (i = 0;i < 40;i++) {

if (((i+1) % 5) != 0) printf("%8i ", fib\_numbers[i]);

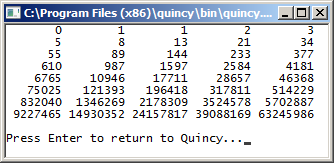
else printf("%8i\n", fib\_numbers[i]);

}

return 0;

}

**Note:** the initialization, condition, and increment are all exactly the same for both loops. It would be easy to include this in the previous loop by copying the if and else statements. I've only kept the loops separate here in order to more precisely fulfill the parameters of the exercise.



2) Write a declaration for a two-dimensional array named *temperature\_readings* that stores one month of hourly temperature readings. (For simplicity, assume that a month has 30 days.) The rows of the array should represent days of the month. The columns should represent hours of the day.

3) Assuming the array of problem 2) contains valid data, use it to construct a program fragment that computes the average temperature for a month (averaged over all days of the month and all hours of the day).

These questions do not specify a source for the temperature information. Instead of just ignoring that, and producing program fragments, I've substituted a random function in order to generate a full program.

The program I've created answers both of the questions, and allowed me to practice and explore a bit more. **I've indicated in caps with comments what parts of the code answer which question.**

What the rand function is replaced with will depend upon the type of system the code will run on. For example, an embedded system with no real-time clock would probably have some kind of one-hour delay function (eg. delay(3600000); for the arduindo IDE), while a computer would probably have a process that runs a file every hour on the hour according to the system time.

Something else I've noticed is that despite the textbook's indication that in an array “...the first index number refers to the row number, whereas the second index number references the column” [1], there seems to be no distinction in the language itself as to which index number corresponds to which dimension. In these simple programs, I can make any index number refer to any dimension, so long as I'm consistent throughout the code. Is this choice actually arbitrary (thus making [row][column] "good practice" for readability) or am I missing something?

There are some formatting issues that I apologize for. In particular, the printf() for the header, and horizontal dividers are quite long.

[1] Stephen G. Kochan, *Programming in C*, 3rd ed., Indianapolis: Sams Publishing, 2005, page 114

/\* Talen Phillips

\* EE107-01

\* 01APR2013

\* "temperature"

\*/

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

/\* I've included define statements to practice the define statement,

\* and to make it easier for someone to edit the code. \*/

#define tot\_days 30

#define tot\_hours 24

int main (void)

{

int hour; //tot\_days and tot\_hours were defined above, so these are just counters

int day;

float sum;

srand (time(NULL));

/\* **ANSWER TO QUESTION 2**: This is the declaration of the array. \*/

float temperature\_matrix[tot\_days][tot\_hours];

/\* This first set of loops fills the array with random numbers from 20.0 to 25.0.

\* The outer loop increments the days, while the inner loop increments the hours

\* and applies each randomly generated number to each position in the array.\*/

for (day=0; day<tot\_days; day++){

for (hour=0; hour<tot\_hours; hour++){

temperature\_matrix[day][hour] = (rand() %50 + 200) \* 0.1;

}

}

/\* This second set of loops prints the entire array to the screen with some nifty

\* formatting that creates a grid that contains the numbers and days and hours.\*/

for (day=0; day<tot\_days; day++){

/\* If we're at the beginning of the array, print the header \*/

if (day==0) printf("TempC || 1am| 2am| 3am| 4am| 5am| 6am| 7am| 8am| 9am|10am| 11am|12am| 1pm| 2pm| 3pm| 4pm| 5pm| 6pm| 7pm| 8pm| 9pm|10pm|11pm|12pm\n ======||====|====|====|====|====|====|====|====|====|====|====|====|====|====|====|====|====|====|====|====|====|====|====|====\n");

/\* If we're not at the beginning of the array, print the horizontal divider. \*/

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

/\* at the beginning of each row, print the day.\*/

printf("%4i |",day+1);

/\* This inner for loop prints the temp at each position in the array\*/

for (hour=0; hour<tot\_hours; hour++) printf("|%4.1f", temperature\_matrix[day][hour]);

/\* at the end of the row, put a newline \*/

putchar('\n');

}

/\* **ANSWER TO QUESTION 3**: this final loop sums all of the values in the array \*/

for (day=0; day<tot\_days; day++){

for (hour=0; hour<tot\_hours; hour++) sum += temperature\_matrix[day][hour];

}

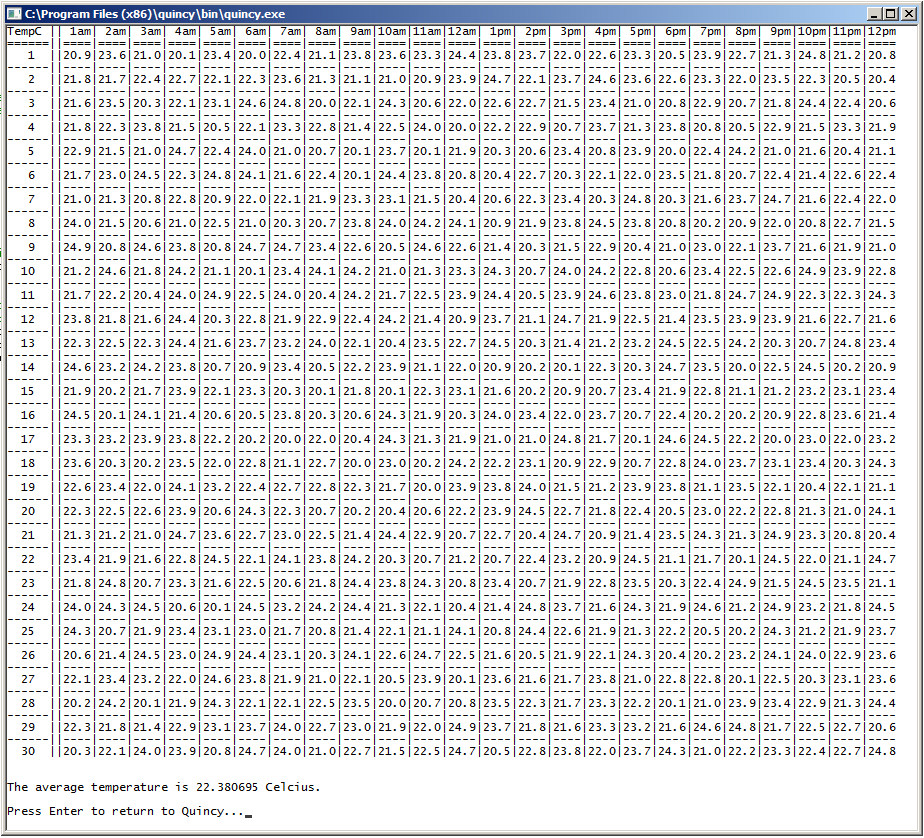
/\* Here we use the sum to find the average of the entire array, and print it \*/

printf("\n\nThe average temperature is %f Celcius.\n",sum/(tot\_days\*tot\_hours));

return 0;

}

Screenshot:



4) Write a program to read in a sentence from the user (maximum 80 characters including punctuation) and writes the sentence to the screen with every word having the first letter capitalized. For example:

Enter a sentence: Hello. I am writing this line!

Hello. I Am Writing This Line!

/\* Talen Phillips

\* EE107-01

\* 01APR2013

\* "Sentence"

\*/

#include <stdio.h>

int main (void)

{

int i;

char sen[81];

printf("Enter a sentence:\n");

/\* The fgets() function is used here because scanf() seems to treat

\* every word as a separate string. \*/

fgets(sen,81,stdin);

/\* The following loop checks each letter, capitalizes it or not

\* according to the condition in the long if statement, and prints

\* it at the end of each iteration. \*/

for (i=0; sen[i]!='\0'; i++){

/\* If the previous character isn't a lower-case letter, capital

\* letter, or apostrophe, AND the current character is a lower

\* case letter, then capitalize it. Otherwise, just print it. \*/

if (((sen[i-1]<'a')||(sen[i-1]>'z'))&&((sen[i-1]<'A')||(sen[i-1]>'Z'))&&(sen[i-1]!='\'')&&(sen[i]>='a')&&(sen[i]<='z'))

putchar(sen[i]-32);

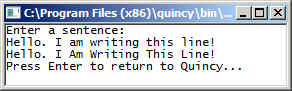
else putchar(sen[i]);

}

return 0;

}

The existence of gets() is a bit confusing, as it just crashes the program when the input overruns the array or whatever other buffer it's assigned to.



5) Write a program to request a first and last name from the user and write to the screen the last name then the initial of the first name separated by a comma and ended with a period. For example:

Enter name: Jim Thorp

Thorp, J.

/\* Talen Phillips

\* EE107-01

\* 01APR2013

\* "Name"

\*/

#include <stdio.h>

int main (void)

{

char first[20];

char last[20];

printf("Enter a name: ");

/\* as mentioned in the answer to question 4, the scanf() function

\* seems to treat each word as a separate string. \*/

scanf("%s %s",first,last);

/\* this loop just prints the last name. **SEE NOTE**! \*/

for (int i=0; last[i]!='\0'; i++) putchar(last[i]);

printf(", ");

/\* this putchar just prints the first letter of the first name. \*/

putchar(first[0]);

putchar('.');

return 0;

}

Note that I've defined i's type within the initialization statement of the for loop. I've done this intentionally to bring up a question:

This seems to work with compilers for C99 and up. It makes the variable local to that one loop. I see a number of potential benefits for this as well as a number of downsides. I was wondering how useful this method might be, and if it's worth using when it's available.

