//

// main.c

// ExtraCredit1

//

// Created by Randy McMillan on 11/2/13.

// Copyright (c) 2013 Randy McMillan. All rights reserved.

//

/\*

\*

\*

\* 1. An integer n is divisible by 9 if the sum of its digits is divisible by 9.

\* Develop a program to display each digit, starting with the rightmost digit. Your program should also determine whether or not the number is divisible by 9. Test it on the following numbers:

\* n = 154368 n = 621594 n = 123456

\* Hint: Use the % operator to get each digit; then use / to remove that digit. So 154368 % 10 gives 8 and 154368 / 10 gives 15436. The next digit extracted should be 6, then 3 and so on.

\*

\*

\* \*/

#include <stdio.h>

#include <math.h>

double input;

int divisible = 1;

void myMain();

double ten = 10;

double numArray[];

double sum = 0;

int sentinel = 0;

int counter = 0;

int cycles = 0;

void calcDivByNine(int);

void printFinalSum(double sum);

int main(int argc, const char \*argv[])

{

myMain();

return 0;

}

void myMain()

{

printf("input number between 0 and 999999999 ---> ");

scanf("%lf", &input);

calcDivByNine(input);

calcDivByNine(sum);

// calcDivByNine(sum);

// printf("input - %lf",input);

printFinalSum(sum);

}

void calcDivByNine(int num)

{

sum = 0;

cycles = cycles + 1;

for (int i = 0; i <= 80; i++) {

numArray[i] = num % 10;

//printf("numArray[%i] = %i\n", i, (int)numArray[i]);

num = num / 10;

//printf("%i\n", num);

counter = counter + 1;

//printf("counter = %i\n", counter);

}

//printf("\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_next for\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

for (int i = counter - 1; i >= 0; i--) {

//printf("\ni = %i\n", i);

//printf("numArray[%i] = %lf\n", i, numArray[i]);

sum = sum + numArray[i];

numArray[i] = 0;

//printf("The sum of the digits = %lf", sum);

}

counter = 0;

}

void printFinalSum(double sum)

{

if (sum == 9||(sum == 0&& cycles>0)) {

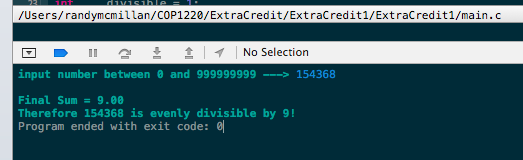
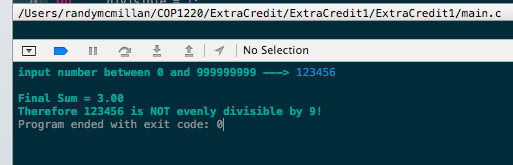
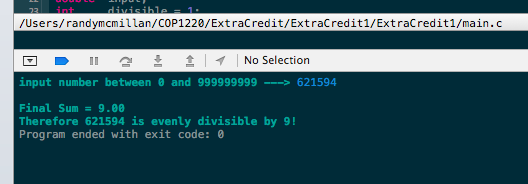
printf("\nFinal Sum = %0.2lf\nTherefore %0.0lf is evenly divisible by 9!\n", sum, input);

} else {

printf("\nFinal Sum = %0.2lf\nTherefore %0.0lf is NOT evenly divisible by 9!\n", sum, input);

}

}



//

// main.c

// ExtraCredit11

//

// Created by Randy McMillan on 11/2/13.

// Copyright (c) 2013 Randy McMillan. All rights reserved.

//

/\*

\*

\*

\* 11. Write a function that will merge the contents of two sorted (ascending order) arrays of type double values, storing the result in an array output parameter (still in ascending order). The function should not assume that both its input parameter arrays are the same length but can assume that one array does not contain two copies of the same value. The result array should also contain no duplicate values.

\* First array

\* 04

\* Second array

\* Result array

\* Hint: When one of the input arrays has been exhausted, do not forget to copy the remaining data in the other array into the result array. Test your function with cases in which (1) the first array is exhausted first, (2) the second array is exhausted first, and (3) the two arrays are exhausted at the same time (i.e., they end with the same value). Remember that the arrays input to this func- tion must already be sorted.

\*

\*

\*/

#include <stdio.h>

double firstArray[] = {-10.5, -1.8, 3.5, 6.3, 7.2};

double secondArray[] = {-1.8, 3.1, 6.3};

double thirdArray[100];

int j = 0;

void sortAndMerge(double \*fA, double \*fB);

int main(int argc, const char \*argv[])

{

for (int i = 0; i != ((sizeof firstArray / sizeof \*firstArray) | (sizeof secondArray / sizeof \*secondArray)); i++) {

printf("\nfirstArray[%i] = %.2lf\n", i, firstArray[i]);

printf("\nsecondArray[%i] = %.2lf\n", i, secondArray[i]);

printf("\nthirdArray[%i] = %lf\n", i, thirdArray[i]);

// double firstArray[] = {-10.5, -1.8, 3.5, 6.3, 7.2};

// double secondArray[] = {-1.8, 3.1, 6.3};

if (firstArray[i] < secondArray[i]) {

thirdArray[j] = firstArray[i];

thirdArray[j] = secondArray[i];

} else {

thirdArray[j] = secondArray[i];

thirdArray[j] = firstArray[i];

}

printf("\nfirstArray[%i] = %.2lf\n", i, firstArray[i]);

printf("\nsecondArray[%i] = %.2lf\n", i, secondArray[i]);

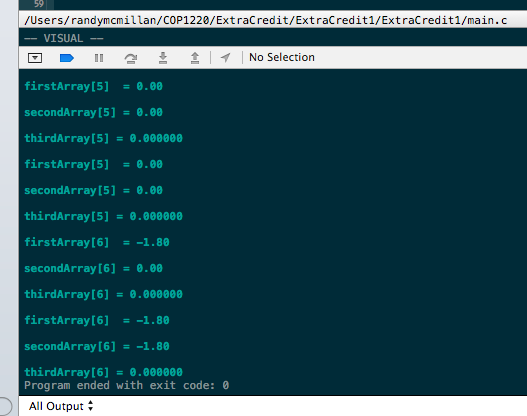
printf("\nthirdArray[%i] = %lf\n", i, thirdArray[i]);

}

return 0;

}

void sortAndMerge(double \*fA, double \*fB) {}



//

// main.c

// ExtraCredit8

//

// Created by Randy McMillan on 11/2/13.

// Copyright (c) 2013 Randy McMillan. All rights reserved.

//

/\*

\*

\* 8. Write a program to process weekly employee time cards for all employees of an organization. Each employee will have three data items: an identification number, the hourly wage rate, and the number of hours worked during a given week. Each employee is to be paid time and a half for all hours worked over 40. A tax amount of 3.625% of gross salary will be deducted. The program out- put should show the employee’s number and net pay. Display the total payroll and the average amount paid at the end of the run.

\*

\*/

#include <stdio.h>

#include <string.h>

#define FORTY 40

#define TAX 3.625

#define ZERO 0

#define LINE 80

int e;

int numberOfEmployees;

int idNum[];

float hourlyRate[];

float hoursWorked[];

char firstName[], lastName[];

char input[LINE];

void inputEmployee();

void myMain();

void inputNumberOfEmployees();

char \*scanLine(char \*dest, int dest\_len);

void assignFirstName(char \*input, int loopLength);

int main(int argc, const char \*argv[])

{

myMain();

return 0;

}

void myMain()

{

inputNumberOfEmployees();

inputEmployee();

}

void inputEmployee()

{

e = e + 1;

for (int i = 0; i < numberOfEmployees; i++) {

//

printf("Please enter the Employees first name --> ");

assignFirstName(scanLine(input, LINE), (int)strlen(input) + 1);

scanLine(input,LINE);

assignFirstName(input, (int)strlen(input)-1);

//scanLine(input, LINE);

//strcpy(&firstName[i], input);

//printf("%s", &firstName[i]);

}

}

void inputNumberOfEmployees()

{

printf("Plese enter the number of employees --> ");

scanf("%i", &numberOfEmployees);

}

void assignFirstName(char \*input, int loopLength) {

//printf("%s",input);

}

// Based on Figure 8.15 in Book

char \*scanLine(char \*dest, int dest\_len)

{

int i, ch;

i = ZERO;

for (ch = getchar(); ch != '\n' && ch != EOF && i < dest\_len - 1; ch = getchar()) {

dest[i++] = ch;

}

dest[i] = '\0';

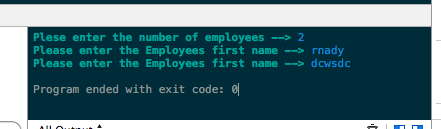
while (ch != '\n' && ch != EOF) {

ch = getchar();

}

return dest;

}



//

// main.c

// ExtraCredit9

//

// Created by Randy McMillan on 11/2/13.

// Copyright (c) 2013 Randy McMillan. All rights reserved.

//

/\*

\*

\*

\* 9. Write an interactive program that plays a game of hangman. Store the word to be guessed in successive elements of an array of individual characters called word. The player must guess the letters belonging to word. The program should terminate when either all letters have been guessed correctly (the player wins) or a specified number of incorrect guesses have been made (the compu- ter wins). Hint: Use another array, guessed, to keep track of the solution so far. Initialize all elements of guessed to the '\*' symbol. Each time a letter in word is guessed, replace the corresponding '\*' in guessed with that letter.

\*

\*

\*/

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <string.h>

// Keywords List 32 words

// <http://tigcc.ticalc.org/doc/keywords.html>

char kwAuto[] = "auto";

char kwBreak[] = "break";

char kwCase[] = "case";

char kwChar[] = "char";

char kwConst[] = "const";

char kwContinue[] = "continue";

char kwDefault[] = "default";

char kwDo[] = "do";

char kwDouble[] = "double";

char kwElse[] = "else";

char kwEnum[] = "enum";

char kwExtern[] = "extern";

char kwFloat[] = "float";

char kwFor[] = "for";

char kwGoto[] = "goto";

char kwIf[] = "if";

char kwInt[] = "int";

char kwLong[] = "long";

char kwRegister[] = "register";

char kwReturn[] = "return";

char kwShort[] = "short";

char kwSigned[] = "signed";

char kwSizeOf[] = "sizeof";

char kwStatic[] = "static";

char kwStruct[] = "struct";

char kwSwitch[] = "switch";

char kwTypeOf[] = "typeof";

char kwUnion[] = "union";

char kwUnSigned[] = "unsigned";

char kwVoid[] = "void";

char kwVolatile[] = "volatile";

char kwWhile[] = "while";

// char dictionary[32][10] = {\*kwAuto,\*kwBreak,\*kwCase,\*kwChar,\*kwConst,\*kwContinue,\*kwDefault,\*kwDo,\*kwDouble,\*kwElse,\*kwEnum,\*kwExtern,\*kwFloat,\*kwFor,\*kwGoto,\*kwIf,\*kwInt,\*kwLong,\*kwRegister,\*kwReturn,\*kwShort,\*kwSigned,\*kwSizeOf,\*kwStatic,\*kwStruct,\*kwSwitch,\*kwTypeOf,\*kwUnion,\*kwUnSigned,\*kwVoid,\*kwVolatile,\*kwWhile};

void populateDictionary();

float returnRand();

float returnClock();

void printKeywordSubs(char word[], int i);

char keyword[100];

int keywordIndex;

int keywordLength;

void setUp();

void myMain();

int main(int argc, const char \*argv[])

{

myMain();

return 0;

}

void myMain()

{

setUp();

}

void setUp()

{

keywordIndex = returnRand();

printf("keywordIndex = %i", keywordIndex);

populateDictionary();

// printf("\nkeyword = %s",&dictionary[keywordIndex]);

printKeywordSubs(keyword, (int)strlen(keyword) - 1);

}

void printKeywordSubs(char word[], int i)

{

for (int i = 0; i < strlen(word); i++) {

printf("\nword[%i] = %c", i, word[i]);

}

}

void populateDictionary()

{

if (keywordIndex == 0) {

strncpy(keyword, kwAuto, strlen(kwAuto));

printf("\nkeyword - %s", &keyword);

}

if (keywordIndex == 1) {

strncpy(keyword, kwBreak, strlen(kwBreak));

printf("\nkeyword - %s", &keyword);

}

if (keywordIndex == 2) {

strncpy(keyword, kwCase, strlen(kwCase));

printf("\nkeyword - %s", &keyword);

}

if (keywordIndex == 3) {

strncpy(keyword, kwChar, strlen(kwChar));

printf("\nkeyword - %s", &keyword);

}

if (keywordIndex == 4) {

strncpy(keyword, kwConst, strlen(kwConst));

printf("\nkeyword - %s", &keyword);

}

if (keywordIndex == 5) {

strncpy(keyword, kwContinue, strlen(kwContinue));

printf("\nkeyword - %s", &keyword);

}

if (keywordIndex == 6) {

strncpy(keyword, kwDefault, strlen(kwDefault));

printf("\nkeyword - %s", &keyword);

}

if (keywordIndex == 7) {

strncpy(keyword, kwDo, strlen(kwDo));

printf("\nkeyword - %s", &keyword);

}

if (keywordIndex == 8) {

strncpy(keyword, kwDouble, strlen(kwDouble));

printf("\nkeyword - %s", &keyword);

}

if (keywordIndex == 9) {

strncpy(keyword, kwElse, strlen(kwElse));

printf("\nkeyword - %s", &keyword);

}

if (keywordIndex == 10) {

strncpy(keyword, kwEnum, strlen(kwEnum));

printf("\nkeyword - %s", &keyword);

}

if (keywordIndex == 11) {

strncpy(keyword, kwExtern, strlen(kwExtern));

printf("\nkeyword - %s", &keyword);

}

if (keywordIndex == 12) {

strncpy(keyword, kwFloat, strlen(kwFloat));

printf("\nkeyword - %s", &keyword);

}

if (keywordIndex == 13) {

strncpy(keyword, kwFor, strlen(kwFor));

printf("\nkeyword - %s", &keyword);

}

if (keywordIndex == 14) {

strncpy(keyword, kwGoto, strlen(kwGoto));

printf("\nkeyword - %s", &keyword);

}

if (keywordIndex == 15) {

strncpy(keyword, kwIf, strlen(kwIf));

printf("\nkeyword - %s", &keyword);

}

if (keywordIndex == 16) {

strncpy(keyword, kwInt, strlen(kwInt));

printf("\nkeyword - %s", &keyword);

}

if (keywordIndex == 17) {

strncpy(keyword, kwLong, strlen(kwLong));

printf("\nkeyword - %s", &keyword);

}

if (keywordIndex == 18) {

strncpy(keyword, kwRegister, strlen(kwRegister));

printf("\nkeyword - %s", &keyword);

}

if (keywordIndex == 19) {

strncpy(keyword, kwReturn, strlen(kwReturn));

printf("\nkeyword - %s", &keyword);

}

if (keywordIndex == 20) {

strncpy(keyword, kwShort, strlen(kwShort));

printf("\nkeyword - %s", &keyword);

}

if (keywordIndex == 21) {

strncpy(keyword, kwSigned, strlen(kwSigned));

printf("\nkeyword - %s", &keyword);

}

if (keywordIndex == 22) {

strncpy(keyword, kwSizeOf, strlen(kwSizeOf));

printf("\nkeyword - %s", &keyword);

}

if (keywordIndex == 23) {

strncpy(keyword, kwStatic, strlen(kwStatic));

printf("\nkeyword - %s", &keyword);

}

if (keywordIndex == 24) {

strncpy(keyword, kwStruct, strlen(kwStruct));

printf("\nkeyword - %s", &keyword);

}

if (keywordIndex == 25) {

strncpy(keyword, kwSwitch, strlen(kwSwitch));

printf("\nkeyword - %s", &keyword);

}

if (keywordIndex == 26) {

strncpy(keyword, kwTypeOf, strlen(kwTypeOf));

printf("\nkeyword - %s", &keyword);

}

if (keywordIndex == 27) {

strncpy(keyword, kwUnion, strlen(kwUnion));

printf("\nkeyword - %s", &keyword);

}

if (keywordIndex == 28) {

strncpy(keyword, kwUnSigned, strlen(kwUnSigned));

printf("\nkeyword - %s", &keyword);

}

if (keywordIndex == 29) {

strncpy(keyword, kwVoid, strlen(kwVoid));

printf("\nkeyword - %s", &keyword);

}

if (keywordIndex == 30) {

strncpy(keyword, kwVolatile, strlen(kwVolatile));

printf("\nkeyword - %s", &keyword);

}

if (keywordIndex == 31) {

strncpy(keyword, kwWhile, strlen(kwWhile));

printf("\nkeyword - %s", &keyword);

}

if (keywordIndex == 32) {

strncpy(keyword, kwAuto, strlen(kwAuto));

printf("\nkeyword - %s", &keyword);

}

if (keywordIndex == 33) {

strncpy(keyword, kwAuto, strlen(kwAuto));

printf("\nkeyword - %s", &keyword);

}

// printf("\n%s",&dictionary[keywordIndex]);

// printf("\n%s\n",&dictionary[keywordIndex]);

}

float returnRand()

{

double clock = returnClock();

clock = clock / 10;

int intpart = (int)clock;

double decpart = clock - intpart;

return decpart \* 10;

}

float returnClock()

{

return clock();

}

