*CSC 141 Introduction to Computer Programming*

Practice Exercises

Hands on Demo

Multidimensional Array

Using Library Functions

forCharacters and strings

code from How to program C 7/e

by Deital chapter -7

Website Reference:

http://www.deitel.com/Books/C/CHowtoProgram7e

Double-sub-array

// Fig. 6.22: fig06\_22.c

// Double-subscripted array manipulations.

#include<stdio.h>

#include<iostream>

Using namespace std;

#define STUDENTS 3

#define EXAMS 4

// function prototypes

int minimum( int grades[][ EXAMS ], size\_t pupils, size\_t tests );

int maximum( int grades[][ EXAMS ], size\_t pupils, size\_t tests );

double average( constint setOfGrades[], size\_t tests );

void printArray( int grades[][ EXAMS ], size\_t pupils, size\_t tests );

// function main begins program execution

int main( void )

{

size\_t student; // student counter

// initialize student grades for three students (rows)

Int studentGrades[ STUDENTS ][ EXAMS ] =

{ { 77, 68, 86, 73 },

{ 96, 87, 89, 78 },

{ 70, 90, 86, 81 } };

// output array studentGrades

puts("The array is:" );

printArray(studentGrades, STUDENTS, EXAMS );

// determine smallest and largest grade values

printf("\n\nLowest grade: %d\nHighest grade: %d\n",

minimum(studentGrades, STUDENTS, EXAMS ),

maximum(studentGrades, STUDENTS, EXAMS ) );

// calculate average grade for each student

for ( student = 0; student < STUDENTS; ++student ) {

printf("The average grade for student %u is %.2f\n",

student, average( studentGrades[ student ], EXAMS ) );

 } // end for

 system("pause");

}// end main

// Find the minimum grade

int minimum( int grades[][ EXAMS ], size\_t pupils, size\_t tests )

{

size\_t i; // student counter

size\_t j; // exam counter

int lowGrade = 100; // initialize to highest possible grade

// loop through rows of grades

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

// loop through columns of grades

for ( j = 0; j < tests; ++j ) {

if ( grades[ i ][ j ] <lowGrade ) {

lowGrade = grades[ i ][ j ];

 } // end if

 } // end inner for

 } // end outer for

Return lowGrade; // return minimum grade

} // end function minimum

// Find the maximum grade

int maximum( int grades[][ EXAMS ], size\_t pupils, size\_t tests )

{

size\_t i; // student counter

size\_t j; // exam counter

int highGrade = 0; // initialize to lowest possible grade

// loop through rows of grades

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

// loop through columns of grades

for ( j = 0; j < tests; ++j ) {

if ( grades[ i ][ j ] > highGrade ) {

highGrade = grades[ i ][ j ];

 } // end if

 } // end inner for

 } // end outer for

return highGrade; // return maximum grade

} // end function maximum

// Determine the average grade for a particular student

double average( constint setOfGrades[], size\_t tests )

{

size\_t i; // exam counter

int total = 0; // sum of test grades

// total all grades for one student

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

total += setOfGrades[ i ];

 } // end for

return ( double ) total / tests; // average

} // end function average

// Print the array

void printArray( int grades[][ EXAMS ], size\_t pupils, size\_t tests )

{

size\_t i; // student counter

size\_t j; // exam counter

// output column heads

printf("%s", " [0] [1] [2] [3]" );

// output grades in tabular format

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

// output label for row

printf("\nstudentGrades[%d] ", i );

// output grades for one student

for ( j = 0; j < tests; ++j ) {

printf("%-5d", grades[ i ][ j ] );

 } // end inner for

 } // end outer for

} // end function printArray

// Fig. 8.2: fig08\_02.c

// Using functions isdigit, isalpha, isalnum, and isxdigit

#include<stdio.h>

#include<ctype.h>

int main( void )

{

printf("%s\n%s%s\n%s%s\n\n", "According to isdigit: ",

isdigit('8' ) ? "8 is a " :"8 is not a ", "digit",

isdigit('#' ) ? "# is a " :"# is not a ", "digit" );

printf("%s\n%s%s\n%s%s\n%s%s\n%s%s\n\n",

"According to isalpha:",

isalpha('A' ) ? "A is a " :"A is not a ", "letter",

isalpha('b' ) ? "b is a " : "b is not a ", "letter",

isalpha('&' ) ? "& is a " :"& is not a ", "letter",

isalpha('4' ) ? "4 is a " :"4 is not a ", "letter" );

printf("%s\n%s%s\n%s%s\n%s%s\n\n",

"According to isalnum:",

isalnum('A' ) ? "A is a " :"A is not a ",

"digit or a letter",

isalnum('8' ) ? "8 is a " :"8 is not a ",

"digit or a letter",

isalnum('#' ) ? "# is a " :"# is not a ",

"digit or a letter" );

printf("%s\n%s%s\n%s%s\n%s%s\n%s%s\n%s%s\n",

"According to isxdigit:",

isxdigit('F' ) ? "F is a " :"F is not a ",

"hexadecimal digit",

isxdigit('J' ) ? "J is a " :"J is not a ",

"hexadecimal digit",

isxdigit('7' ) ? "7 is a " :"7 is not a ",

"hexadecimal digit",

isxdigit('$' ) ? "$ is a " :"$ is not a ",

"hexadecimal digit",

isxdigit('f' ) ? "f is a " : "f is not a ",

"hexadecimal digit" );

} // end main

// Fig. 8.3: fig08\_03.c

// Using functions islower, isupper, tolower, toupper

#include<stdio.h>

#include<ctype.h>

int main( void )

{

printf("%s\n%s%s\n%s%s\n%s%s\n%s%s\n\n",

"According to islower:",

islower('p' ) ? "p is a " : "p is not a ",

"lowercase letter",

islower('P' ) ? "P is a " :"P is not a ",

"lowercase letter",

islower('5' ) ? "5 is a " :"5 is not a ",

"lowercase letter",

islower('!' ) ? "! is a " : "! is not a ",

"lowercase letter" );

printf("%s\n%s%s\n%s%s\n%s%s\n%s%s\n\n",

"According to isupper:",

isupper('D' ) ? "D is an " :"D is not an ",

"uppercase letter",

isupper('d' ) ? "d is an " : "d is not an ",

"uppercase letter",

isupper('8' ) ? "8 is an " :"8 is not an ",

"uppercase letter",

isupper('$' ) ? "$ is an " :"$ is not an ",

"uppercase letter" );

printf("%s%c\n%s%c\n%s%c\n%s%c\n",

"u converted to uppercase is ", toupper( 'u' ),

"7 converted to uppercase is ", toupper('7' ),

"$ converted to uppercase is ", toupper('$' ),

"L converted to lowercase is ", tolower('L' ) );

} // end main

// Fig. 8.4: fig08\_04.c

// Using functions isspace, iscntrl, ispunct, isprint, isgraph

#include<stdio.h>

#include<ctype.h>

int main( void )

{

printf("%s\n%s%s%s\n%s%s%s\n%s%s\n\n",

"According to isspace:",

"Newline", isspace('\n' ) ? " is a " : " is not a ",

"whitespace character", "Horizontal tab",

isspace('\t' ) ? " is a " : " is not a ",

"whitespace character",

isspace('%' ) ? "% is a " :"% is not a ",

"whitespace character" );

printf("%s\n%s%s%s\n%s%s\n\n", "According to iscntrl:",

"Newline", iscntrl('\n' ) ? " is a " : " is not a ",

"control character", iscntrl( '$' ) ? "$ is a " :

"$ is not a ", "control character" );

printf("%s\n%s%s\n%s%s\n%s%s\n\n",

"According to ispunct:",

ispunct(';' ) ? "; is a " :"; is not a ",

"punctuation character",

ispunct('Y' ) ? "Y is a " :"Y is not a ",

"punctuation character",

ispunct('#' ) ? "# is a " :"# is not a ",

"punctuation character" );

printf("%s\n%s%s\n%s%s%s\n\n", "According to isprint:",

isprint('$' ) ? "$ is a " :"$ is not a ",

"printing character",

"Alert", isprint('\a' ) ? " is a " : " is not a ",

"printing character" );

printf("%s\n%s%s\n%s%s%s\n", "According to isgraph:",

isgraph('Q' ) ? "Q is a " :"Q is not a ",

"printing character other than a space",

"Space", isgraph(' ' ) ? " is a " : " is not a ",

"printing character other than a space" );

} // end main

/ Fig. 8.6: fig08\_06.c

// Using function strtod

#include<stdio.h>

#include<stdlib.h>

int main( void )

{

// initialize string pointer

const char \*string = "51.2% are admitted"; // initialize string

double d; // variable to hold converted sequence

char \*stringPtr; // create char pointer

 d = strtod( string, &stringPtr );

printf("The string \"%s\" is converted to the\n", string );

printf("double value %.2f and the string \"%s\"\n", d, stringPtr );

} // end main

// Fig. 8.7: fig08\_07.c

// Using function strtol

#include<stdio.h>

#include<stdlib.h>

int main( void )

{

const char \*string = "-1234567abc"; // initialize string pointer

char \*remainderPtr; // create char pointer

long x; // variable to hold converted sequence

 x = strtol( string, &remainderPtr, 0 );

printf("%s\"%s\"\n%s%ld\n%s\"%s\"\n%s%ld\n",

"The original string is ", string,

"The converted value is ", x,

"The remainder of the original string is ",

remainderPtr,

"The converted value plus 567 is ", x + 567 );

} // end main

// Fig. 8.22: fig08\_22.c

// Using function strpbrk

#include<stdio.h>

#include<string.h>

int main( void )

{

const char \*string1 = "This is a test"; // initialize char pointer

const char \*string2 = "beware"; // initialize char pointer

printf("%s\"%s\"\n'%c'%s\n\"%s\"\n",

"Of the characters in ", string2,

 \*strpbrk( string1, string2 ),

" appears earliest in ", string1 );

} // end main

// Fig. 8.10: fig08\_10.c

// Using functions fgets and putchar

#include<stdio.h>

#define SIZE 80

void reverse( const char \* const sPtr ); // prototype

int main( void )

{

char sentence[ SIZE ]; // create char array

puts("Enter a line of text:" );

// use fgets to read line of text

fgets( sentence, SIZE, stdin );

puts("\nThe line printed backward is:" );

reverse( sentence );

} // end main

// recursively outputs characters in string in reverse order

void reverse( const char \* const sPtr )

{

// if end of the string

if ( '\0' == sPtr[ 0 ] ) { // base case

return;

 } // end if

else { // if not end of the string

reverse(&sPtr[ 1 ] ); // recursion step

putchar(sPtr[ 0 ] ); // use putchar to display character

 } // end else

} // end function reverse

// Fig. 8.11: fig08\_11.c

// Using function getchar.

#include<stdio.h>

#define SIZE 80

int main( void )

{

int c; // variable to hold character input by user

char sentence[ SIZE ]; // create char array

int i = 0; // initialize counter i

// prompt user to enter line of text

puts("Enter a line of text:" );

// use getchar to read each character

while ( i < SIZE - 1 && ( c = getchar() ) != '\n' ) {

sentence[ i++ ] = c;

 } // end while

sentence[ i ] = '\0'; // terminate string

// use puts to display sentence

puts("\nThe line entered was:" );

puts( sentence );

} // end main

// Fig. 8.12: fig08\_12.c

// Using function sprintf

#include<stdio.h>

#define SIZE 80

int main( void )

{

char s[ SIZE ]; // create char array

int x; // x value to be input

double y; // y value to be input

puts("Enter an integer and a double:" );

scanf("%d%lf", &x, &y );

sprintf( s, "integer:%6d\ndouble:%8.2f", x, y );

printf("%s\n%s\n",

"The formatted output stored in array s is:", s );

} // end main

// Fig. 8.13: fig08\_16.c

// Using function sscanf

#include<stdio.h>

int main( void )

{

char s[] = "31298 87.375"; // initialize array s

int x; // x value to be input

double y; // y value to be input

sscanf( s, "%d%lf", &x, &y );

printf("%s\n%s%6d\n%s%8.3f\n",

"The values stored in character array s are:",

"integer:", x, "double:", y );

} // end main

// Fig. 8.15: fig08\_15.c

// Using functions strcpy and strncpy

#include<stdio.h>

#include<string.h>

#define SIZE1 25

#define SIZE2 15

int main( void )

{

char x[] = "Happy Birthday to You"; // initialize char array x

char y[ SIZE1 ]; // create char array y

char z[ SIZE2 ]; // create char array z

// copy contents of x into y

printf("%s%s\n%s%s\n",

"The string in array x is: ", x,

"The string in array y is: ",strcpy( y, x ) );

// copy first 14 characters of x into z. Does not copy null

// character

strncpy( z, x, SIZE2 - 1 );

z[ SIZE2 - 1 ] = '\0'; // terminate string in z

printf("The string in array z is: %s\n", z );

} // end main

// Fig. 8.16: fig08\_16.c

// Using functions strcat and strncat

#include<stdio.h>

#include<string.h>

int main( void )

{

char s1[ 20 ] = "Happy "; // initialize char array s1

char s2[] = "New Year "; // initialize char array s2

char s3[ 40 ] = ""; // initialize char array s3 to empty

printf("s1 = %s\ns2 = %s\n", s1, s2 );

// concatenate s2 to s1

printf( "strcat( s1, s2 ) = %s\n", strcat( s1, s2 ) );

// concatenate first 6 characters of s1 to s3. Place '\0'

// after last character

printf( "strncat( s3, s1, 6 ) = %s\n", strncat( s3, s1, 6 ) );

// concatenate s1 to s3

printf( "strcat( s3, s1 ) = %s\n", strcat( s3, s1 ) );

} // end main

// Fig. 8.18: fig08\_18.c

// Using functions strcmp and strncmp

#include<stdio.h>

#include<string.h>

int main( void )

{

const char \*s1 = "Happy New Year"; // initialize char pointer

const char \*s2 = "Happy New Year"; // initialize char pointer

const char \*s3 = "Happy Holidays"; // initialize char pointer

printf("%s%s\n%s%s\n%s%s\n\n%s%2d\n%s%2d\n%s%2d\n\n",

"s1 = ", s1, "s2 = ", s2, "s3 = ", s3,

"strcmp(s1, s2) = ", strcmp( s1, s2 ),

"strcmp(s1, s3) = ", strcmp( s1, s3 ),

"strcmp(s3, s1) = ", strcmp( s3, s1 ) );

printf("%s%2d\n%s%2d\n%s%2d\n",

"strncmp(s1, s3, 6) = ", strncmp( s1, s3, 6 ),

"strncmp(s1, s3, 7) = ", strncmp( s1, s3, 7 ),

"strncmp(s3, s1, 7) = ", strncmp( s3, s1, 7 ) );

} // end main

// Fig. 8.20: fig08\_20.c

// Using function strchr

#include<stdio.h>

#include<string.h>

int main( void )

{

const char \*string = "This is a test"; // initialize char pointer

char character1 = 'a'; // initialize character1

char character2 = 'z'; // initialize character2

// if character1 was found in string

if ( strchr( string, character1 ) != NULL ) {

printf("\'%c\' was found in \"%s\".\n",

 character1, string );

 } // end if

else { // if character1 was not found

printf("\'%c\' was not found in \"%s\".\n",

 character1, string );

 } // end else

// if character2 was found in string

if ( strchr( string, character2 ) != NULL ) {

printf("\'%c\' was found in \"%s\".\n",

 character2, string );

 } // end if

else { // if character2 was not found

printf("\'%c\' was not found in \"%s\".\n",

 character2, string );

 } // end else

system("pause")

} // end main

// Fig. 8.25: fig08\_25.c

// Using function strstr

#include<stdio.h>

#include<string.h>

int main( void )

{

const char \*string1 = "abcdefabcdef"; // string to search

const char \*string2 = "def"; // string to search for

printf("%s%s\n%s%s\n\n%s\n%s%s\n",

"string1 = ", string1, "string2 = ", string2,

"The remainder of string1 beginning with the",

"first occurrence of string2 is: ",

strstr( string1, string2 ) );

} // end main

// Fig. 8.35: fig08\_35.c

// Using function strlen

#include<stdio.h>

#include<string.h>

int main( void )

{

// initialize 3 char pointers

const char \*string1 = "abcdefghijklmnopqrstuvwxyz";

const char \*string2 = "four";

const char \*string3 = "Boston";

printf("%s\"%s\"%s%u\n%s\"%s\"%s%u\n%s\"%s\"%s%u\n",

"The length of ", string1, " is ", strlen( string1 ),

"The length of ", string2, " is ", strlen( string2 ),

"The length of ", string3, " is ", strlen( string3 ) );

} // end main