*CSC 141 Introduction to Computer Programming*

Practice Exercises

Hands on Demo

Structures

Unions, Bitwise operators

Enumeration Constants

Demo Lecture 27

Examples from How to program C 7/e

byDeital chapter -10

Website Reference:

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

1. Addressing-stru-members

// Fig. 10.2: fig10\_02.c

// Structure member operator and

// structure pointer operator

#include<stdio.h>

#include<iostream>

// card structure definition

struct card {

char \*face; // define pointer face

char \*suit; // define pointer suit

}; // end structure card

int main( void )

{

struct card aCard; // define one struct card variable

struct card \*cardPtr; // define a pointer to a struct card

// place strings into aCard

aCard.face = "Ace";

aCard.suit = "Spades";

cardPtr = &aCard; // assign address of aCard to cardPtr

printf("%s%s%s\n%s%s%s\n%s%s%s\n", aCard.face, " of ", aCard.suit,

cardPtr->face, " of ", cardPtr->suit,

( \*cardPtr ).face, " of ", ( \*cardPtr ).suit );

system("pause");

} // end main

1. Card-shuff-and-deal using structures

// Fig. 10.3: fig10\_03.c

// Card shuffling and dealing program using structures

#include<stdio.h>

#include<stdlib.h>

#include<time.h>

#include<iostream>

#define CARDS 52

#define FACES 13

// card structure definition

struct card {

constchar \*face; // define pointer face

constchar \*suit; // define pointer suit "Hearts", "Diamonds", "Clubs", "Spades"

}; // end struct card

typedefstruct card Card; // new type name for struct card

// prototypes

voidfillDeck( Card \* constwDeck, constchar \* wFace[], constchar \* wSuit[] );

void shuffle( Card \* constwDeck );

void deal( const Card \* constwDeck );

int main( void )

{

Card deck[ CARDS ]; // define array of Cards

// initialize array of pointers

constchar \*face[] = { "Ace", "Deuce", "Three", "Four", "Five",

"Six", "Seven", "Eight", "Nine", "Ten",

"Jack", "Queen", "King"};

// initialize array of pointers

constchar \*suit[] = { "Hearts", "Diamonds", "Clubs", "Spades"};

srand( time( NULL ) ); // randomize

fillDeck( deck, face, suit ); // load the deck with Cards

shuffle( deck ); // put Cards in random order

deal( deck ); // deal all 52 Cards

system("pause");

} // end main

// place strings into Card structures

voidfillDeck( Card \* constwDeck, constchar \* wFace[], constchar \* wSuit[] )

{

size\_t i; // counter

// loop through wDeck

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

wDeck[ i ].face = wFace[ i % FACES ];

// (0,1,2,3,4,5,6,7,8,9,10,11,12)%13 = 0,1,2,3,4,5,6,7,8,9,10,11,12

// (13,14,15,16,17,18,19,20,21,22,23,24,25)%13 = 0,1,2,3,4,5,6,7,8,9,10,11,12

// (26,27,28,29,30,31,32,33,34,35,36,37,38)%13 = 0,1,2,3,4,5,6,7,8,9,10,11,12

// (39,40,41,42,43,44,45,46,47,48,49,50,51)%13 = 0,1,2,3,4,5,6,7,8,9,10,11,12

//"Ace", "Deuce", "Three", "Four", "Five", //"Six", "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King"

wDeck[ i ].suit = wSuit[ i / FACES ];

// (0,1,2,3,4,5,6,7,8,9,10,11,12)/13 = 0) Hearts,Hearts,Hearts,Hearts,Hearts,Hearts,Hearts,........

// (13,14,15,16,17,18,19,20,21,22,23,24,25)/13 = 1) Diamods,Diamods,Diamods,Diamods,Diamods,Diamods,Diamods,Diamods,....

// (26,27,28,29,30,31,32,33,34,35,36,37,38)/13 = 2) Clubs,Clubs,Clubs,Clubs,Clubs,Clubs,Clubs,Clubs,............

// (39,40,41,42,43,44,45,46,47,48,49,50,51)/13 = 3) Spades,Spades,Spades,Spades,Spades,Spades,Spades,............

} // end for

} // end function fillDeck

// shuffle cards

void shuffle( Card \* constwDeck )

{

size\_t i; // counter

size\_t j; // variable to hold random value between 0 - 51

Card temp; // define temporary structure for swapping Cards

// loop through wDeck randomly swapping Cards

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

j = rand() % CARDS;

temp = wDeck[ i ];

wDeck[ i ] = wDeck[ j ];

wDeck[ j ] = temp;

} // end for

} // end function shuffle

// deal cards

void deal( const Card \* constwDeck )

{

size\_t i; // counter

// loop through wDeck

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

printf("%5s of %-8s%s", wDeck[ i ].face, wDeck[ i ].suit,

( i + 1 ) % 4 ?" " :"\n" );

} // end for

} // end function deal

1. unions

// Fig. 10.5: fig10\_05.c

// Displaying the value of a union in both member data types

#include<stdio.h>

#include<iostream>

// number union definition

union number {

int x;

double y;

}; // end union number

int main( void )

{

union number value; // define union variable

value.x = 100; // put an integer into the union

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

"Put 100 in the integer member",

"and print both members.",

"int:", value.x,

"double:", value.y );

value.y = 100.0; // put a double into the same union

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

"Put 100.0 in the floating member",

"and print both members.",

"int:", value.x,

"double:", value.y );

system("pause");

} // end main

4.Bit-manipulation

// Fig. 10.7: fig10\_07.c

// Displaying an unsigned int in bits

#include<stdio.h>

#include<iostream>

voiddisplayBits( unsignedint value ); // prototype

int main( void )

{

unsignedint x; // variable to hold user input

printf("%s", "Enter a nonnegative int: " );

scanf("%u", &x );

displayBits( x );

} // end main

// display bits of an unsigned int value

voiddisplayBits( unsignedint value )

{

unsignedint c; // counter

// define displayMask and left shift 31 bits

unsignedintdisplayMask = 1 << 31;

printf("%10u = ", value );

// loop through bits

for ( c = 1; c <= 32; ++c ) {

putchar( value &displayMask ? '1' :'0' );

value<<= 1; // shift value left by 1

if ( c % 8 == 0 ) { // output space after 8 bits

putchar(' ' );

} // end if

} // end for

putchar('\n' );

system("pause");

} // end function displayBits

5.Bitwise and and or

// Fig. 10.9: fig10\_09.c

// Using the bitwise AND, bitwise inclusive OR, bitwise

// exclusive OR and bitwise complement operators

#include<stdio.h>

#include<iostream>

void displayBits( unsignedint value ); // prototype

int main( void )

{

unsigned int number1;

unsigned int number2;

unsigned int mask;

unsigned int setBits;

// demonstrate bitwise AND (&)

number1 = 65535;

mask = 1;

puts("The result of combining the following" );

displayBits( number1 );

displayBits( mask );

puts("using the bitwise AND operator & is" );

displayBits( number1 & mask );

// demonstrate bitwise inclusive OR (|)

number1 = 15;

setBits = 241;

puts("\nThe result of combining the following" );

displayBits( number1 );

displayBits(setBits );

puts("using the bitwise inclusive OR operator | is" );

displayBits( number1 | setBits );

// demonstrate bitwise exclusive OR (^)

number1 = 139;

number2 = 199;

puts("\nThe result of combining the following" );

displayBits( number1 );

displayBits( number2 );

puts("using the bitwise exclusive OR operator ^ is" );

displayBits( number1 ^ number2 );

// demonstrate bitwise complement (~)

number1 = 21845;

puts("\nThe one's complement of" );

displayBits( number1 );

puts("is" );

displayBits( ~number1 );

system("pause");

} // end main

// display bits of an unsigned int value

void displayBits( unsignedint value )

{

unsigned int c; // counter

// declare displayMask and left shift 31 bits

unsignedintdisplayMask = 1 << 31;

printf("%10u = ", value );

// loop through bits

for ( c = 1; c <= 32; ++c ) {

putchar( value &displayMask ? '1' :'0' );

value<<= 1; // shift value left by 1

if ( c % 8 == 0 ) { // output a space after 8 bits

putchar(' ' );

} // end if

} // end for

putchar('\n' );

} // end function displayBits

6. Bitwise shift

// Fig. 10.13: fig10\_13.c

// Using the bitwise shift operators

#include<stdio.h>

voiddisplayBits( unsignedint value ); // prototype

int main( void )

{

unsignedint number1 = 960; // initialize number1

// demonstrate bitwise left shift

puts("\nThe result of left shifting" );

displayBits( number1 );

puts("8 bit positions using the left shift operator << is" );

displayBits( number1 << 8 );

// demonstrate bitwise right shift

puts("\nThe result of right shifting" );

displayBits( number1 );

puts("8 bit positions using the right shift operator >> is" );

displayBits( number1 >> 8 );

} // end main

// display bits of an unsigned int value

voiddisplayBits( unsignedint value )

{

unsignedint c; // counter

// declare displayMask and left shift 31 bits

unsignedintdisplayMask = 1 << 31;

printf("%7u = ", value );

// loop through bits

for ( c = 1; c <= 32; ++c ) {

putchar( value &displayMask ? '1' :'0' );

value<<= 1; // shift value left by 1

if ( c % 8 == 0 ) { // output a space after 8 bits

putchar(' ' );

} // end if

} // end for

putchar('\n' );

} // end function displayBits

7. card shuffling using bit fields

// Fig. 10.16: fig10\_16.c

// Representing cards with bit fields in a struct

#include<stdio.h>

#define CARDS 52

// bitCard structure definition with bit fields

structbitCard {

unsignedint face : 4; // 4 bits; 0-15

unsignedint suit : 2; // 2 bits; 0-3

unsignedint color : 1; // 1 bit; 0-1

}; // end structbitCard

typedefstructbitCard Card; // new type name for structbitCard

voidfillDeck( Card \* constwDeck ); // prototype

void deal( const Card \* constwDeck ); // prototype

int main( void )

{

Card deck[ CARDS ]; // create array of Cards

fillDeck( deck );

deal( deck );

} // end main

// initialize Cards

voidfillDeck( Card \* constwDeck )

{

size\_t i; // counter

// loop through wDeck

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

wDeck[ i ].face = i % (CARDS / 4);

wDeck[ i ].suit = i / (CARDS / 4);

wDeck[ i ].color = i / (CARDS / 2);

} // end for

} // end function fillDeck

// output cards in two-column format; cards 0-25 subscripted with

// k1 (column 1); cards 26-51 subscripted with k2 (column 2)

void deal( const Card \* constwDeck )

{

size\_t k1; // subscripts 0-25

size\_tk2; // subscripts 26-51

// loop through wDeck

for ( k1 = 0, k2 = k1 + 26; k1 < CARDS / 2; ++k1, ++k2 ) {

printf("Card:%3d Suit:%2d Color:%2d ",

wDeck[ k1 ].face, wDeck[ k1 ].suit, wDeck[ k1 ].color );

printf("Card:%3d Suit:%2d Color:%2d\n",

wDeck[ k2 ].face, wDeck[ k2 ].suit, wDeck[ k2 ].color );

} // end for

} // end function deal

8. enumerated

// Fig. 10.18: fig10\_18.c

// Using an enumeration

#include<stdio.h>

// enumeration constants represent months of the year

enum months {

JAN = 1, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC

}; // end enum months

int main( void )

{

enum months month; // can contain any of the 12 months

// initialize array of pointers

constchar \*monthName[] = { "", "January", "February", "March",

"April", "May", "June", "July", "August", "September", "October",

"November", "December" };

// loop through months

for ( month = JAN; month <= DEC; ++month ) {

printf("%2d%11s\n", month, monthName[ month ] );

} // end for

} // end main

Skip

// ex10\_16.c

// This program determines whether a value is a multiple of X.

#include<stdio.h>

int multiple( intnum ); // prototype

int main( void )

{

int y; // y will hold an integer entered by the user

puts("Enter an integer between 1 and 32000: " );

scanf("%d", &y );

// if y is a multiple of X

if ( multiple( y ) ) {

printf("%d is a multiple of X\n", y );

} // end if

else {

printf("%d is not a multiple of X\n", y );

} // end else

} // end main

// determine whether num is a multiple of X

int multiple( intnum )

{

int i; // counter

int mask = 1; // initialize mask

intmult = 1; // initialize mult

for ( i = 1; i <= 10; ++i, mask <<= 1 ) {

if ( ( num& mask ) != 0 ) {

mult = 0;

break;

} // end if

} // end for

returnmult;

} // end function multiple

9. Mystery

// ex10\_17.c

#include<stdio.h>

#include <iostream>

int mystery( unsignedint bits ); // prototype

int main( void )

{

unsignedint x; // x will hold an integer entered by the user

puts("Enter an integer: " );

scanf("%u", &x );

printf("The result is %d\n", mystery( x ) );

system("pause");

} // end main

// What does this function do?

int mystery( unsignedint bits )

{

unsignedint i; // counter

unsignedint mask = 1 << 31; // initialize mask

unsignedint total = 0; // initialize total

for ( i = 1; i <= 32; ++i, bits <<= 1 ) {

if ( ( bits & mask ) == mask ) {

++total;

} // end if

} // end for

return !( total % 2 ) ? 1 : 0;

} // end function mystery