#include "Array.h"

//Constructor

Array::Array(int sz)

{

//Allocate pointer

p = new int[sz];

//Set size

size = sz;

}

//Destructor

Array::~Array()

{

//Cleanup memory

delete[] p;

}

//Copy Constructor

/\*Array::Array(const Array &a)

{

//Variable Declaration/Initialization

int i = 0;

//Make an exact copy of a!

p = new int[a.size];

size = a.size;

//copy elements one at a time

for (i = 0; i < a.size; i++)

p[i] = a.p[i];

}\*/

//put method

void Array::put(int i, int j)

{

if ((i >= 0) && (i < size))

p[i] = j;

}

//get method

int Array::get(int i)

{

return(p[i]);

}

--------------------------------------------------------------/////-----------------------------------------------------

#include <iostream>

#include <ctime>

#include "graph1.h"

//Prototypes

void shuffle(int\* deck, int no\_cards);

void displayHand(int\* hand, int no\_cards,int x, int y);

void sort(int\* a, int no\_ele);

int binarySearch(int \*list, int no\_of\_ele, int target);

int main()

{

//Variable Declaration/Initialization

int i = 0;

int x = 0;

int y = 0;

int no\_cards = 0;

int search\_card = 0;

char repeat = 'y';

int index = -1;

const int TOTAL\_CARDS = 13;

const int CAPACITY = 13;

int\* deck = new int[TOTAL\_CARDS];

int\* hand = new int[CAPACITY];

//Display Grpaphics

displayGraphics();

//Shuffle the deck

shuffle(deck, TOTAL\_CARDS);

cout << "Enter # of cards to deal: ";

cin >> no\_cards;

for (i = 0; i < no\_cards; i++)

{

hand[i] = deck[i];

}

//Sort the hand

sort(hand,no\_cards);

//Display the sorted deck

displayHand(hand,no\_cards,150,50);

//Search for cards

do

{

//Prompt for a card to search for

cout << "Enter # of card to search for: ";

cin >> search\_card;

//Perform binary search

index = binarySearch(hand,no\_cards,search\_card);

//Test the search

if (index == -1)

{

cout << "Card not found" << endl;

}

else

{

cout << "Card found" << endl;

}

cout << "Repeat? (y/n): ";

cin >> repeat;

}while((repeat == 'y') || (repeat == 'Y') );

return 0;

}

void sort(int\* a, int no\_ele)

{

//Variable Declaration/Initialization

int n = 0;

int k = 0;

int sm\_index = 0;

int smallest = 0;

int temp1 = 0;

int temp2 = 0;

for(k=0; k < no\_ele -1; k++) // size-1 = number of passes

{

smallest=a[k];

sm\_index=k;

for(n= k+1; n < no\_ele; n++) // size = # elem. to look at

{

if(a[n] < smallest)

{

smallest=a[n];

sm\_index=n;

}

}

if (smallest < a[k])

{

temp1=a[k];

a[k]=a[sm\_index];

a[sm\_index] = temp1;

}

}

}

void shuffle(int\* deck, int no\_cards)

{

int i = 0;

int j = 0;

bool duplicate = false;

//Set the seed

srand(time(0));

//Generate no\_colors random numbers

for (i = 0; i < no\_cards; i++)

{

deck[i] = (rand()%no\_cards)+2;

//Check all previous values

do

{

//Set duplicate to false

duplicate = false;

//Check prvious values

for (j = 0; j < i; j++)

{

if (deck[i] == deck[j])

{

deck[i] = (rand()%no\_cards)+2;

duplicate = true;

}

}

}while(duplicate);

}

}

void displayHand(int\* hand, int no\_cards,int x, int y)

{

//Work with only hearts

int i = 0;

string name;

for (i = 0; i < no\_cards; i++)

{

//Use sprintf to combine name info

name = "h" + to\_string(hand[i]) + ".bmp";

displayBMP(name,x,y);

//Offset card

y+= 20;

}

}

int binarySearch(int \*list, int no\_of\_ele, int target)

{

//Variable Declaration/Initialization

int low = 0;

int high = 0;

int middle = 0;

bool found = false;

int index = -1;

//Initialize low/high

low = 0;

high = no\_of\_ele - 1;

while((low <= high) && (!found))

{

middle = (low + high)/2;

if (list[middle] == target)

found = true;

else if (target > list[middle])

low = middle + 1;

else

high = middle -1;

}

if (!found)

{

index = -1;

}

else

{

index = middle;

}

return(index);

}

----------------------------------------------------/////----------------------------------------------------------

#include <iostream>

#include <ctime>

#include "graph1.h"

class Cards

{

private:

int noCards;

int topCard;

int\* deck;

public:

Cards();

Cards(int noCards);

~Cards();

int getNoCards();

int dealACard();

void shuffle();

void displayCard(int cardNo, int x, int y);

};

Cards::Cards(int noCards)

{

this->noCards = noCards;

topCard = 0;

deck = new int[noCards];

}

Cards::~Cards()

{

delete[] deck;

}

int Cards::getNoCards()

{

return noCards;

}

int Cards::dealACard()

{

if (topCard < noCards)

{

return(deck[topCard++]);

}

else

{

//Indicate that the deck has been dealt by returning -1

return -1;

}

}

void Cards::shuffle()

{

int i = 0;

int j = 0;

bool duplicates = false;

srand(time(0));

for (i = 0; i < noCards; i++)

{

deck[i] = rand()%noCards + 2;

do

{

//Check for duplicate

duplicates = false;

for (j = 0; j < i; j++)

{

if (deck[j] == deck[i])

{

deck[i] = rand()%noCards + 2;

duplicates = true;

}

}

}while(duplicates);

}

}

void Cards::displayCard(int cardNo, int x, int y)

{

//Display hearts for this case

string fn = "h" + to\_string(cardNo) + ".bmp";

displayBMP(fn, x,y);

}

int main()

{

//Only process one suit

int i = 0;

int x = 100;

int y = 50;

int cardNo = 0;

Cards cards(13);

displayGraphics();

//Shuffle the cards

cards.shuffle();

//Deal the cards (one at a time)

for (i = 0; i < cards.getNoCards(); i++)

{

//Get the card no from the top

cardNo = cards.dealACard();

//Display the card

cards.displayCard(cardNo,x,y);

Sleep(1000);

//Increment y

y += 20;

}

return 0;

}

--------------------------------------------------------////--------------------------------------------------

#include <iostream>

#include "Array.h"

using namespace std;

void function(Array a,int size);

int main()

{

//Create an array of size 10

int i = 0;

int size = 10;

Array array1(size);

Array array2; //What happens here ?

//Initialize the array

for (i = 0; i < size; i++)

{

array1.put(i,i);

}

//Call function foo

function(array1,size);

//display array1

for (i = 0; i < size; i++)

{

cout << "Element no " << i << " " << array1.get(i) << endl;

}

return 0;

}

void function(Array a,int size)

{

//Variable Declaration/Initialization

int i = 0;

int total = 0;

//Calculate total

for (i = 0; i < size; i++)

{

total = total + a.get(i);

}

return;

}

-------------------------------------/////-----------------------------------

// Class definition in "Date.h"

#include <iostream>

#include "Date.h"

using namespace std;

Date::Date::Date()

{

month = 0;

day = 0;

year = 0;

}

Date::Date(int month, int day, int year)

{

this->month = month;

this->day = day;

this->year = year;

}

void Date::setDate(int month, int day, int year)

{

this->month = month;

this->day = day;

this->year = year;

}

int Date::getDay()

{

return day;

}

int Date::getMonth()

{

return month;

}

int Date::getYear()

{

return year;

}

//-----------------------------------------------------------------------

// Performs assignment of dates

//

void Date::operator=(Date right\_side)

{

this->month = right\_side.month;

this->day = right\_side.day;

this->year = right\_side.year;

}

//-----------------------------------------------------------------------

// Overloaded operator for adding dates

//

//

//

Date Date::operator+(Date val)

{

Date result;

//Add days together first

result.day = (day + val.day)%31;

result.month = (month + val.month)%12;

result.year = year + val.year;

return(result);

}

Date Date::operator-(Date val)

{

Date result;

//Eliminates overflow problem

result.day = abs(day - val.day);

result.month = abs(month - val.month);

result.year = abs(year - val.year);

return(result);

}

Date Date::operator+(int day)

{

//Add days together

Date result;

result.day = this->day + day;

result.month = month;

result.year = year;

return(result);

}

//-----------------------------------------------------------------------

// Increments the date (postfix operator)

//

Date Date::operator++(int dummy)

{

dummy = 0; // to prevent compiler warning

// Save old value of date

Date old\_value = \*this;

// Increment date (call prefix operator to do the work)

++(\*this);

// Return old value

return old\_value;

}

//-----------------------------------------------------------------------

// Increments the date (prefix operator)

//

Date& Date::operator++( )

{

day++;

// Return new value of date

return \*this;

}

//-----------------------------------------------------------------------

// Decrements the date (postfix operator)

//

Date Date::operator--(int dummy)

{

dummy = 0; // to prevent compiler warning

// Save old value of date

Date old\_value = \*this;

// Decrement date (call prefix operator to do the work)

--(\*this);

// Return old value

return old\_value;

}

//-----------------------------------------------------------------------

// Decrements the date (prefix operator)

//

Date& Date::operator--( )

{

day--;

// Return new value of date

return \*this;

}

//-----------------------------------------------------------------------

// Increments this date by the specified day\_count

//

void Date::operator+=(int day\_count)

{

// Call prefix operator the number of times specified by day\_count

for (int count = 1; count <= day\_count; count++)

{

++(\*this);

}

}

Date& Date::returnRef(Date& c1)

{

c1.day++;

c1.month++;

c1.year++;

return c1;

}

-----------------------------------------------------/////---------------------------------------------------------

#include <iostream>

#include <string>

#include "Employee.h"

using namespace std;

//Static data field initialization

int Employee::count = 0;

const double Employee::pi = 3.14159268;

//Create employee object

Employee::Employee(int salary)

{

//Set salary

this->salary = salary;

id = count++;

}

Employee::~Employee()

{

//Decrement counter

count--;

cout << "Decrementing Employee count: " << endl;

}

int Employee::getId()

{

return id;

}

int Employee::getSalary()

{

return salary;

int count = getCount();

}

int Employee::getCount()

{

//this->id = 9;

//Cannot access private data fields

//return (id) - this won't work

return count;

}

-----------------------------------------------------/////---------------------------------------------------------

#include <iostream>

#include "Employee.h"

using namespace std;

int main()

{

//Can access public static methods without instantiating an object

cout << "Current Employee Count" << Employee::getCount() << endl;

//What occurs here?

Employee \*employee1 = new Employee(23000);

Employee \*employee2 = new Employee(33000);

//What is displayed here?

cout << Employee::getCount() << endl;

//What occurs here?

delete employee1;

//What is displayed here?

cout << Employee::getCount() << endl;

return 0;

}

-----------------------------------------------------/////---------------------------------------------------------

#include <iostream>

#include <cmath>

#include "graph1.h"

#include "GenLine.h"

GenLine::GenLine()

{

p1.setPoint(0,0);

p2.setPoint(0,0);

desc = "";

}

void GenLine::setDesc(string desc)

{

this->desc = desc;

}

string GenLine::getDesc()

{

return(desc);

}

void GenLine::setLine(GenPoint p1, GenPoint p2)

{

//Set the point

this->p1.setPoint(p1.getX(), p1.getY());

this->p2.setPoint(p2.getX(), p2.getY());

}

GenPoint GenLine::getPoint1()

{

return(p1);

}

GenPoint GenLine::getPoint2()

{

return(p2);

}

double GenLine::getLength()

{

double distance = sqrt( pow( (double)p1.getX() - (double)p2.getX(),2.0) + pow((double)p1.getY()-(double)p2.getY(),2.0));

return(distance);

}

void GenLine::draw()

{

//draw the line

int obj\_no = drawLine(p1.getX(), p1.getY(), p2.getX(), p2.getY(), 1);

//Draw purple circles at end points

obj\_no = drawCircle(5,p1.getX(),p1.getY());

setColor(obj\_no,255,0,255);

//Draw purple circles at end points

obj\_no = drawCircle(5,p2.getX(),p2.getY());

setColor(obj\_no,255,0,255);

//Draw purple circle at midpint

GenPoint mid = getMidPoint();

obj\_no = drawCircle(5,mid.getX(), mid.getY());

setColor(obj\_no,255,0,255);

}

GenPoint GenLine::getMidPoint()

{

GenPoint mid;

GenPoint p1 = this->getPoint1();

GenPoint p2 = this->getPoint2();

mid.setPoint( (p1.getX() + p2.getX())/2.0,(p1.getY() + p2.getY())/2.0);

return(mid);

}

-----------------------------------------------------/////---------------------------------------------------------

#include <iostream>

#include "GenPoint.h"

void GenPoint::setPoint(int x, int y)

{

this->x = x;

this->y = y;

}

int GenPoint::getX()

{

return x;

}

int GenPoint::getY()

{

return y;

}

-----------------------------------------------------/////---------------------------------------------------------

#include <iostream>

#include "graph1.h"

#include "GenPolygon.h"

GenPolygon::GenPolygon()

{

//Set to nulls

this->lines = NULL;

this->no\_lines = 3;

}

GenPolygon::GenPolygon(int no\_lines)

{

//Set based on no\_lines

this->no\_lines = no\_lines;

this->lines = new GenLine[no\_lines];

}

GenPolygon::~GenPolygon()

{

//Cleanup

delete[] lines;

}

GenPolygon::GenPolygon(const GenPolygon& poly)

{

//Make a copy of poly to this!

int i = 0;

//Copy no\_lines first

this->no\_lines = poly.no\_lines;

//Allocate lines for this

this->lines = new GenLine[this->no\_lines];

//Set each line one at a time

for (i = 0; i < this->no\_lines; i++)

{

//Set each line for this!

this->setLineAt(i,poly.lines[i]);

}

}

int GenPolygon::getNoLines()

{

return(no\_lines);

}

void GenPolygon::setNoLines(int no\_lines)

{

//Clip values

if (no\_lines <= 0)

{

no\_lines = 3;

}

else if (no\_lines > 10)

{

no\_lines = 10;

}

//Set the lines

this->no\_lines = no\_lines;

}

GenLine GenPolygon::getLineAt(int index)

{

//Return the line at the index

GenLine line;

line.setLine(this->lines[index].getPoint1(), this->lines[index].getPoint2());

return line;

}

void GenPolygon::setLineAt(int index, GenLine line)

{

//Set this line points

this->lines[index].setLine(line.getPoint1(), line.getPoint2());

//Set the description

this->lines[index].setDesc(line.getDesc());

}

void GenPolygon::draw()

{

//Draw the lines

int i = 0;

//Draw each line

for (i = 0; i < no\_lines; i++)

{

//Draw the line

this->lines[i].draw();

}

}

-----------------------------------------------------/////---------------------------------------------------------

#include <iostream>

#include <cstring>

#include <cstdlib>

#include "GenString1.h"

using namespace std;

//-----------------------------------------------------------------------

// Default Constructor: Initializes string to null (0)

//

GenString::GenString()

{

string = NULL;

}

//-----------------------------------------------------------------------

// Constructor: Calls setString( )

//

GenString::GenString( char \*string\_value)

{

string = NULL; // Initialize string before calling setString()

setString( string\_value);

}

//-----------------------------------------------------------------------

// Copy Constructor

//

GenString::GenString( const GenString &string\_value)

{

string = 0; // Initialize string before calling setString()

setString( string\_value.string);

}

//-----------------------------------------------------------------------

// Destructor: frees new memory

//

GenString::~GenString()

{

delete[] string;

}

//-----------------------------------------------------------------------

// Sets the string to string\_value

//

void GenString::setString(char \*string\_value)

{

// Check for Previous Allocation and deallocate if necessary

delete [] string;

// Allocate string

string = new char[strlen(string\_value) + 1]; // enough space for '\0'

// Copy string, including null

strcpy\_s( string, strlen(string) +1, string\_value);

}

//-----------------------------------------------------------------------

// Returns the string

//

char\* GenString::getString()

{

// Alternative to constant: Return a copy of the string

int str\_length = strlen(string);

// Allocate copy

char \*copy = new char[str\_length + 1]; // enough space for '\0'

// Copy string, including null

strcpy\_s(copy, str\_length + 1,string);

return copy;

}

//-----------------------------------------------------------------------

// Returns true if string equals another\_string

//

bool GenString::isEqual( GenString another\_string)

{

if (strcmp(string,another\_string.getString()) == 0)

return(true);

else

return(false);

}

//-----------------------------------------------------------------------

// Converts string to lowercase

//

void GenString::toLower()

{

for (int index = 0; index < (int)strlen(string); index++)

{

string[index] = tolower(string[index]);

}

}

//-----------------------------------------------------------------------

// Converts string to uppercase

//

void GenString::toUpper()

{

for (int index = 0; index < (int)strlen(string); index++)

{

string[index] = toupper(string[index]);

}

}

//-----------------------------------------------------------------------

// Returns a character of the string at the specified index.

// If the index is < 0 or >= length, prints an error message

//

char GenString::getChar( int index)

{

if (index < 0 || index >= (int)strlen(string))

{

cout << "\n\nERROR: Index is out-of-bounds.\n\n";

return ('\0');

}

return string[index];

}

const char\* GenString::getString1()

{

return (string);

}

void GenString::change() const

{

//string = new char[10];

}

//-----------------------------------------------------------------------

// Returns the character at the specified index.

// If the index is < 0 or > length, prints an error message

// (referenced object[] is an l-value)

//

char& GenString::operator[](int index)

{

// Check range

if (index < 0 || index > (int)strlen(string))

{

cout << "\n\nERROR: Index is out-of\_bounds.\n";

exit(1);

}

return string[(int)index];

}

//-----------------------------------------------------------------------

// Returns the character at the specified index.

// If the index is < 0 or > length, prints an error message

// (referenced object[] is an l-value)

//

/\*const char& GenString::operator[](int index) const

{

// Check range

if (index < 0 || index > strlen(string))

{

cout << "\n\nERROR: Index is out-of\_bounds.\n";

exit(1);

}

return string[index];

}\*/

//-----------------------------------------------------------------------

-----------------------------------------------------/////---------------------------------------------------------

#include <iostream>

#include <iomanip>

#include "GenString1.h"

using namespace std;

// Driver function to test Class String member functions

int main( void )

{

// Variable declarations/initializations

const int SIZE = 5;

int i = 0;

int index = 0;

// \*\*\* Test the default constructor, setString() and getString()

// Specify an array of strings with the default constructor

GenString string\_array[SIZE];

GenString gen;

//Test parameterized array

GenString string\_array1[SIZE] = {GenString("Joe Smith"), GenString("Jim Davis"),GenString("joe smith")};

GenString string\_array2[] = {GenString("Jim Johnson"), GenString(), GenString("Joe Smith"), GenString("Jerry Jones")};

GenString gen\_string\_extra("Joe Smith");

const char \*string3 = gen\_string\_extra.getString1();

//string3[0] = 'J';

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*Additional Tests showing use of Pointers\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//Allocate space to hold the pointer

GenString \*gen\_string\_ptr;

//Set gen\_string\_ptr to 3rd element (element of 2) of the array

gen\_string\_ptr = &string\_array2[2];

cout << (\*gen\_string\_ptr).getString() << endl;

cout << gen\_string\_ptr->getString() << endl;

cout << gen\_string\_ptr[0].getString() << endl;

cout << gen\_string\_ptr[1].getString() << endl;

cout << (gen\_string\_ptr + 1)->getString() << endl;

//Allocate gen\_string\_ptr to a single pointer (use parametrized constructor

gen\_string\_ptr = new GenString;

delete gen\_string\_ptr;

gen\_string\_ptr = new GenString();

delete gen\_string\_ptr;

gen\_string\_ptr = new GenString(" The brown fox ran quickly ");

//Cleanup the memory (destructor called automatically)

delete gen\_string\_ptr;

return 0;

}

-----------------------------------------------------/////---------------------------------------------------------

#include <iostream>

#include <ctime>

#include "graph1.h"

using namespace std;

//Prototypes

void insertSort (int\* a, int no\_ele);

void insertSortCard(int\* a, int no\_ele, int card\_no);

void shuffle(int\* deck, int no\_cards);

void displayHand(int\* deck, int no\_cards,int x, int y);

int main()

{

int i = 0;

int x = 0;

int y = 0;

const int TOTAL\_CARDS = 13;

int no\_cards = 6;

int\* deck = new int[TOTAL\_CARDS];

int\* hand = new int[no\_cards];

string fn;

//Display Grpaphics

displayGraphics();

//Shuffle the deck

shuffle(deck, TOTAL\_CARDS);

//Display the hand

displayHand(deck,TOTAL\_CARDS,50,50);

Sleep(3000);

//Perform insertion sort on the deck

insertSort(deck,TOTAL\_CARDS);

//Display the hand

displayHand(deck,TOTAL\_CARDS,150,50);

Sleep(2000);

//Reshuffle

shuffle(deck, TOTAL\_CARDS);

//Maintain sorted list

for (i = 0; i < no\_cards; i++)

{

//Display Card to be inserted

fn = "h" + to\_string(deck[i]) + ".bmp";

displayBMP(fn,500,0);

Sleep(3000);

//Insert the card

insertSortCard(hand,i,deck[i]);

displayHand(hand,i+1,250,50);

}

return 0;

}

void shuffle(int\* deck, int no\_cards)

{

int i = 0;

int j = 0;

bool duplicate = false;

//Set the seed

srand(time(0));

//Generate no\_colors random numbers

for (i = 0; i < no\_cards; i++)

{

deck[i] = (rand()%no\_cards)+2;

//Check all previous values

do

{

//Set duplicate to false

duplicate = false;

//Check prvious values

for (j = 0; j < i; j++)

{

if (deck[i] == deck[j])

{

deck[i] = (rand()%no\_cards)+2;

duplicate = true;

}

}

}while(duplicate);

}

}

void insertSort (int\* a, int n)

{

//Variable Declaration/Initialization

int i = 0;

int j = 0;

int v = 0;

// For each element after the first:

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

{

// Look back through the previous (sorted) elts.

// Insert this elt after first that is smaller,

// moving the others up as you go by them.

v = a[i];

for (j = i-1; j >= 0 && a[j] > v; j--)

{

// a[j] is bigger; move it up

a[j+1] = a[j];

}

// a[j] <= v; put v after a[j]

a[j+1] = v;

}

}

void displayHand(int\* deck, int no\_cards,int x, int y)

{

//Work with only hearts

int i = 0;

string name;

for (i = 0; i < no\_cards; i++)

{

//Use sprintf to combine name info

name = "h" + to\_string(deck[i]) + ".bmp";

displayBMP(name,x,y);

//Offset card

y+= 20;

}

}

void insertSortCard(int\* a, int no\_ele, int card\_no)

{

//Variable Declaration/Initialization

int v = 0;

int j = 0;

v = card\_no;

for (j = no\_ele-1; j >= 0 && a[j] > v; j--)

{

// a[j] is bigger; move it up

a[j+1] = a[j];

}

// a[j] <= v; put v after a[j]

a[j+1] = v;

}

-----------------------------------------------------/////---------------------------------------------------------

#include <iostream>

#include "graph1.h"

using namespace std;

//Class Declaration

class Employee

{

private: //These are the private data field

int id;

int dept\_id;

int yrs\_svc;

public:

Employee();

Employee(int id\_val, int dept\_id\_val, int yrs\_svc);

void setId(int id);

void setDeptId(int dept\_id);

int getId();

void setYrsSvc(int yrs\_svc);

int getYrsSvc();

bool detSvcAward();

};

Employee::Employee(int id\_val, int dept\_id\_val, int yrs\_svc\_val )

{

setId(id\_val);

setDeptId(dept\_id\_val);

setYrsSvc(yrs\_svc\_val);

}

Employee::Employee()

{

id = 0;

dept\_id = 0;

yrs\_svc = 0;

}

//Class Implementation

void Employee::setId(int id\_val)

{

id = id\_val;

}

void Employee::setDeptId(int dept\_id\_val)

{

dept\_id = dept\_id\_val;

}

void Employee::setYrsSvc(int yrs\_svc\_val)

{

yrs\_svc = yrs\_svc\_val;

}

int Employee::getYrsSvc()

{

//Date date = hire\_date - curr\_date;

//Convert date to int no\_yrs

//int no\_yrs

return(yrs\_svc);

}

//Implement all other setters

//Implement a getter for id

int Employee::getId()

{

return(id);

}

bool Employee::detSvcAward()

{

if (yrs\_svc > 5)

return(true);

else

return(false);

}

int main()

{

//Variable Declaration/Initialization

Employee emp1;

Employee emp2(1807,21,3);

int id = 0;

int id1 = 0;

//Set the id for emp1

emp1.setId(1806);

//Retrieve the id for emp1/emp2

id = emp1.getId();

id1 = emp2.getId();

return 0;

}

-----------------------------------------------------/////---------------------------------------------------------

#include <iostream>

#include <ctime>

#include <conio.h>

#include "graph1.h"

//Prototypes

void shuffle(int\* deck, int no\_cards);

void displayDeck(int\* deck, int no\_cards,int x, int y);

void sort(int\* a, int no\_ele);

int main()

{

//Variable Declaration/Initialization

int i = 0;

int x = 0;

int y = 0;

char repeat = 'y';

const int TOTAL\_CARDS = 13;

int\* deck = new int[TOTAL\_CARDS];

//Display Grpaphics

displayGraphics();

//Shuffle the deck

shuffle(deck, TOTAL\_CARDS);

//Display the Deck

displayDeck(deck,TOTAL\_CARDS,50,50);

cout << "Hit y to continue: ";

cin >> repeat;

//Sort the cards

sort(deck,TOTAL\_CARDS);

//Display the deck

displayDeck(deck,TOTAL\_CARDS,150,50);

return 0;

}

void sort(int\* a, int no\_ele)

{

//Variable Declaration/Initialization

int n = 0;

int k = 0;

int sm\_index = 0;

int smallest = 0;

int temp1 = 0;

int temp2 = 0;

for(k=0; k < no\_ele -1; k++) // size-1 = number of passes

{

smallest=a[k];

sm\_index=k;

for(n= k+1; n < no\_ele; n++) // size = # elem. to look at

{

if(a[n] < smallest)

{

smallest=a[n];

sm\_index=n;

}

}

if (smallest < a[k])

{

temp1=a[k];

a[k]=a[sm\_index];

a[sm\_index] = temp1;

}

}

}

void shuffle(int\* deck, int no\_cards)

{

int i = 0;

int j = 0;

bool duplicate = false;

//Set the seed

srand(time(0));

//Generate no\_colors random numbers

for (i = 0; i < no\_cards; i++)

{

deck[i] = (rand()%no\_cards)+2;

//Check all previous values

do

{

//Set duplicate to false

duplicate = false;

//Check prvious values

for (j = 0; j < i; j++)

{

if (deck[i] == deck[j])

{

deck[i] = (rand()%no\_cards)+2;

duplicate = true;

}

}

}while(duplicate);

}

}

void displayDeck(int\* deck, int no\_cards,int x, int y)

{

//Work with only hearts

int i = 0;

string name;

for (i = 0; i < no\_cards; i++)

{

//Use sprintf to combine name info

name = "h" + to\_string(deck[i]) + ".bmp";

displayBMP(name,x,y);

//Offset card

y+= 20;

}

}

-----------------------------------------------------/////---------------------------------------------------------

#include <iostream>

#include <cstdlib>

#include "graph1.h"

#include "Date.h"

using namespace std;

Date operator+(double value, Date c);

ostream& operator<<(ostream &out, Date right\_side);

istream& operator>>(istream &in, Date& right\_side);

Gout& operator<<(Gout& out, Date& right\_side);

int main()

{

int a = 9;

Date c1(0,0,0);

Date c2(1,1,1);

Date c3(2,2,2);

Date c4;

Date c5;

Date c6(3,3,3);

Date c7(4,4,4);

Date c8(5,5,5);

Date& c9 = c3;

Date& c13 = c5;

c9 = c6;

//Display graphics

displayGraphics();

c4 = c1 + c2;

c4 = ((c1 - c2) + c3)+a;

c5 = (c1 + c2)-c3;

c4 = c1++ + c2;

c4 = ++c1 + c2;

c5 = c2-c3++ + ++c6+c7;

Date c10 = 9.0 + c4;

Date& c11 = c2.returnRef(c3);

((c3.returnRef(c3)).returnRef(c3)).returnRef(c3);

//cout << c11 << endl;

cout << &c11 << endl;

cout.operator<<(9.0).operator<<("Joe").operator<<(endl);

c4.operator+(c5);

int aa = 9;

system("cls");

cin >> c8 >> c4 >> aa;

cout << c1 << c2 << c3 << endl;

//Display date using gout

gout << c9 << endg;

return 0;

}

Date operator+(double value, Date c)

{

Date result;

result.setDate(c.getDay() + (int)value,c.getMonth(),c.getYear());

return(result);

}

ostream& operator<<(ostream &out, Date right\_side)

{

out << "Day: " << right\_side.day << " " << "Month: " << right\_side.month << endl;

return (out);

}

istream& operator>>(istream &in, Date& right\_side)

{

int day;

int month;

int year;

cout << "Enter day,month,year:";

in >> day >> month >> year;

right\_side.day = day;

right\_side.month = month;

right\_side.year = year;

return(in);

}

Gout& operator<<(Gout& out, Date& right\_side)

{

out << setPos(100,100) << "Day: " << right\_side.getDay() << " Month: " << right\_side.getMonth()

<< "Year: " << right\_side.getYear() << endg;

return(out);

}

-----------------------------------------------------/////---------------------------------------------------------

#include <iostream>

#include "graph1.h"

#include "GenPoint.h"

#include "GenLine.h"

#include "GenPolygon.h"

using namespace std;

//Pass by value

void printPolygon(GenPolygon poly);

int main ()

{

//Variable Declaration/Initialization

int i = 0;

int x = 0;

int y = 0;

string info;

char repeat = 'y';

GenPoint p1;

GenPoint p2;

GenLine line;

GenPolygon poly(4); //Set to Quadrilateral

//Display graphics

displayGraphics();

//Set the line description to "Vertical"

line.setDesc("Vertical");

do

{

//Prompt for Point 1

cout << "Enter x/y coords for Point 1: ";

cin >> x >> y;

//Set the point 1

p1.setPoint(x,y);

//Prompt for Point 2

cout << "Enter x/y coords for Point 2: ";

cin >> x >> y;

//Set the point 2

p2.setPoint(x,y);

//Set the line

line.setLine(p1,p2);

//Draw the line/circular endpoints

line.draw();

//Display the length at Point1

gout << setPos(line.getPoint1().getX()+15,line.getPoint1().getY() + 12) << "Length: " << line.getLength() << endg;

//Display the midpoint at Midpoint

gout << setPos(line.getMidPoint().getX() + 15, line.getMidPoint().getY() ) << "Mid Point: (" <<

line.getMidPoint().getX() << "," << line.getMidPoint().getY() << ")" << endg;

//Display info at p2

gout << setPos(line.getPoint2().getX() + 15, line.getPoint2().getY()) << "Info: " << line.getDesc() << endg;

//Process polygon info

cout << endl << endl << endl;

cout << "Processing Polygon Info" << endl << endl;

//Create Polygon from no\_lines

for (i = 0; i < poly.getNoLines(); i++)

{

//Prompt for points for each line

//Prompt for Point 1

cout << "Enter x/y coords for Point 1: ";

cin >> x >> y;

//Set Point 1

p1.setPoint(x,y);

//Prompt for Point 2

cout << "Enter x/y coords for Point 2: ";

cin >> x >> y;

//Set Point 2

p2.setPoint(x,y);

//Set the line

line.setLine(p1,p2);

//Set the desc

info += "Line #";

info += to\_string(i);

line.setDesc(info);

//Set the Polygon

poly.setLineAt(i,line);

}

//Display info

printPolygon(poly);

//Display polygon

poly.draw();

printPolygon(poly);

//Repeat?

cout << "Repeat program? (y/n): ";

cin >> repeat;

//Clear graphics/console

system("cls");

clearGraphics();

}while( (repeat == 'y') || (repeat == 'Y') );

return 0;

}

void printPolygon(GenPolygon poly)

{

//Variable Declaration/Initialization

int i = 0;

//Get the lines

//Print the info to the console

for (i = 0; i < poly.getNoLines(); i++)

{

cout << poly.getLineAt(i).getPoint1().getX() << "," << poly.getLineAt(i).getPoint1().getY() << endl;

}

}