By Section – Grouped Function Document

-The goal is to build a reference list of the different ways to perform similar functions:

Potential Include Headers:

#include <iostream>

#include "graph1.h"

#include <ctime>

#include <cstdlib>

#include <cstring>

#include <iomanip>

#include <cmath>

#include <string>

#include "Array.h"

#include "Date.h"

#include "Employee.h"

#include "GenString1.h"

#include "GenPoint.h"

#include "GenLine.h"

#include "GenPolygon.h"

#include "GenString1.h"

#include <conio.h>

Constructors AND Overloaded Constructors:

The implicit this pointer is used only **within** the method. It refers to the address of the object that the method is invoked on. The “this” pointer is used for resolving ambiguity - the data field preceded by this-> is always the private data field.

Multiple methods within a class can have the same method name too. Overloaded methods consist of same name, different parameters/number of parameters.

Consider the constructors for the Rectangle class:

Rectangle();

Rectangle(int w, int h);

Overloaded methods (for an angle class) might look like this:

void setAngle(double angle\_in\_deg);

void setAngle(float angle\_in\_deg);

void setAngle(double radians, int factor);

Constructor & Overloaded Constructor Examples:

//Constructor

Array::Array(int sz)

{

//Allocate pointer

p = new int[sz];

//Set size

size = sz;

}

--<><><><><><><><><><><><><><>--

Copy Constructor (Used with dynamically allocated Arrays)

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];

}

--<><><><><><><><><><><><><><>--

Cards::Cards(int noCards)

{

this->noCards = noCards;

topCard = 0;

deck = new int[noCards];

}

Destructors:

Destructor is necessary for cleaning up dynamically allocated pointers within constructors. An example of this is: **deck = new int[noCards];** We clean dynamically allocated pointer up using an unloaded Cards() destructor or, from earlier, an unloaded Array() destructor.

Cards::~Cards()

{

delete[] deck;

}

Array::~Array()

{

delete[] p;

}

VOID Setters and Getters:

Add a Super Basic setSomething example:

When accessing an object’s members (functions) – You must use the object’s name with a “.” And then the member (function) name with the appropriate arguments. Direct examples from our labs and in-class examples:

void GenLine::setDesc(string desc)

{

this->desc = desc;

}

string GenLine::getDesc()

{

return(desc);

}

In this example, the method - void GenLine::setDesc(string desc) – passes a locally defined variable “desc” to the function parameter “desc” which can be used elsewhere in the program. The this-> designator specifies that specific “desc” as the local version of “desc” as compared to the methods “desc” parameter.

Some more advanced implementations of Void setters:

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

{

//Set the point

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

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

}

-In this example, we received the X/Y in the GenPoint::setPoint(int x, int y) method from MAIN.

-We stored those values in the locally defined X and Y of the setPoint method.

-WITHIN the GenLine::setLine method, we define local variables p1 and p2.

-In the GenLine function header, we have identified our parameters as (GenPoint p1, and GenPoint p2).

-This allows us to reach into the GenPoint object to use the setPoint function and “get” the X/Y data.

>> setPoint has the values for X/Y, but it’s the getX and getY functions that can return the values.

>> p1 and p2 are locally defined variables, BUT they are also labeled with the GenPoint prefix as that’s where the data originally came from.

VOID Getters with CONST

Functions declared in the class (.h) with the “const” keyword will also have “const” in the function header located in the (.cpp) file. All “getData()” functions should include the const keyword.