1.1 Multiple Choice

```
#include <iostream>
#include <string>
#include <vector>
#include <fstream>
using namespace std;

class Holodeck{
public:
Holodeck(int val):program(val){}
void display() {cout<<"Holodeck programs: "<<program<<endl;}
private:
  int program = 17;
};
class Galaxy{
public:
  void display() {cout << "Galaxy class ship\n";}
private:
  Holodeck something;
};
int main(){
  Galaxy gal;
  gal.display();
}
```

What is the result of compiling and running the code?

Question 9

```
string& add_chars(string& str){
  str += "mto";
  return str;
}

int main(){
  string orig;
  string upd=add_chars(orig);

  for (size_t i =0; i<orig.size(); i++){
    cout<<orig[i]<<' ';
  }
}
```

What will be the result of compiling and running the program?

Given the program below, what would be the result of compiling and executing the code? Completely fill the circle next to your choice.

```
double* update(const double& num){
      const double* db1_ptr  = &num;
      cout<<*db1_ptr<<' ';
      db1_ptr = new double(num*2);
      return db1_ptr;
}

int main(){
      double val = -2.3;
      double* result_ptr = update(val);
      cout<< *result_ptr << ' ';
      delete result_ptr;
}
```

```
class Building{
public:
    int get_room_count() const { return num_rooms;}
private:
    int num_rooms;
};

int main(){
    Building bldg;
    cout<< bldg.get_room_count();
}
```

Question 3

Consider the building class defined above. If a function prototype for a function named print is declared as follows,

void print(Building* bldg_ptr);

Which expression can invoke the get_room_count method on the Building object “pointed at” by blldg_ptr in the print function’s definition? Completely fill the circle next to your choice.

Question 4
Consider the existence of a class named Paint. In a program, two Pain objects have been instantiated by the statements below:

Paint pnt1(“red”);
Paint pnt2(“blue”)

After the objects are created, a subsequent line in the program’s source code contains the following:

pnt 1 = pnt 2;

The statement above produces which of the following function calls?

```
pnt1.operator = (pnt2)
```

1.2 Copy Constructor

1.2.1

Consider the following definition of the PictureFrame class

```
class PictureFrame{
public:
        PictureFrame (double l, double w, double h): length(l), width(w), height(h) {pic = new
        Picture;}
        ~PictureFrame() {delete pic;}
private:
        double length, width, height;
        Picture* pic;
};
```

Assume that a definition of the Picture class exists and includes a copy constructor. Implement the copy constructor for the PictureFrame class such that it could be called as follows:

PictureFrame frame2(frame1);

```
PictureFrame(const PictureFrame &rhs): length(rhs.length),
width(rhs.width),height(rhs.height){
      pic = new Picture(*rhs.pic)
}
```

1.2.1
Given the following definition of the class Coffee,

```
class Coffee{
private:
        string blend
        double* ounces;
};
```

Write a copy constructor for the class such that it could be called as:

Coffee coffee2(coffee1);

```
Coffee(const Coffee& c){
        blend = c.blend;
        ounces = new double(*c.ounces);
}
```

1.3 Ranged For

A program includes the declaration of a vector as follows:

vector<string> user_names;

During the course of the program, the vector is populated with string values. Each string in the user_names vector must have an underscore character added to the end of the string. Write a ranged for (also known as the “foreach” loop) that modifies the strings in user_names such that each string in the vector will have an underscore appended to it.

```
for (char c: user_names){
      user_names + “_”;

}
```

Using ranged for (also known as “foreach”) display the vector of ints called intVec

```
for (int i: intVec){
        cout<< intVec[i] <<endl;
}
```

Consider a program defining a class named Pirate. An overloaded output operator function exists that accepts a Pirate instance as its second parameter. The program creates a vector of Pirate objects named pirates:

vector<Pirate> pirates;

The vector is populated with Pirate objects. Write a ranged for loop that outputs each Pirate on a separate line

```
for (Pirate p: pirates)
{
      p.print();
}
```

1.4 Struct Questions: Fill, Clear and Total

1. 4.1

```
struct Pizza{
        int size;
        string topping;
        double price;
};
```

fillData function

```
void fillData(ifstream& ifs, vector<Pizza>& inventory)
{
      Pizza p;
      while(ifs >> p.size>> p.topping>>p.price)
      {
          inventory.push_back(p);
      }
}
```

filter function

```
void filter(const vector<pizza> &inventory, double limit)
{
        cout<< “Pizzas having price greater that ” << limit <<;
        for (Pizza pizza:inventor)
        {
                if(pizza.price >= limit)
                {
                   cout<< “Size: ” << pizza.size << “\n”;
                   cout<< “Topping:” << pizza.topping << “\n”;
                   cout<< “Price: $” << pizza.price << “\n\n”;
                }
        }
}
```

```
struct Thing {
	vector<int> stuff;
}
```

write the following two functions,

fill: fills a vector with data from a file stream.
totalStuff: Passed a vector of Things. Computed and returns a single int which is the total of all the ints in all of things in the vector

Part A

```
void fill(ifstream& ifs, vector<Thing>& things)
{
	int n, data;
	while(ifs >> n)
	{
		Thing thing; //new Thing
		for(int i=0; i<n; i++) //filling the Thing
		{
			ifs >> data;
			thing.stuff.push_back(data);
		}
		things.push_back(thing); //adding Thing to vector
	}
}
```

Part B

```
int totalStuff(vector<Thing> things)
{
	int total=0;
	for(int i=0; i<things.size(); i++)
	{
		for(int j=0; j<things[i].stuff.size(); j++)
		total+=things[i].stuff[j]; //updating total
	}
	return total;
}
```

Given the type Thing:

```
struct Thing{
        string foo;
        int bar;
};
```

write the two functions:

fill: fills the vector with data read from an input stream.The lines of the file each have a string and an integer to be used for the foo and bar field respectively. You may assume the stream has been correctly opened.

release: frees up all the memory that was allocated and zeroing the size of the vector (NOT DYNAMIC)

Part A

```
void fill(ifstream& ifs, vector<Thing>& things){
      string foo;
      int bar;
      while(ifs >> foo>> bar ){
            Thing newThing;
            newThing.foo = foo;
            newThing.bar = bar;
            things.push_back(newthing);
}
```

Part B

```
void release(vector<Thing>& things){
      vector.clear()
}
```

```
struct Account{
	int acc_id;
	int balance;
	Account(int id, int amnt=0): acc_id(id), balance(amnt){}
}
```

Create a function that reads through a file and uses the contents of the file to modify an array that holds a maximum capacity of say 10 pointers to Account objects.

Part A

```
void read_accounts(ifstream& ifs, Account** accnts){
	int id, bal;
	string blank1, blank2;
	size_t i=0;
	while(ifs >> blank1>>id>>blank2>>bal & i <10){
		accnts[i] = new Account(id,bal);
		i +=1;
	}
}
```

Write a function that then iterates over this array of Account pointers and frees up all the memory within the array including the array itself.

Part B

```
void clear_accounts(Account** accnts){
	size_t i =0;
	while(i<10) {
		delete accnts[i];
		accnts[i]=nullptr;
		i+=1;
	}
	delete[ ] accnts;
}
```

Question 10

Given the following definition of a Block, complete the 4 sub-questions below. For each part, assume that the code will be included in the program's main function.

```
struct Block{
      char label;
};
```

Part A.
Define a local variable data that is a vector of Block pointers. The pointers will be added in the next part.

```
vector<Block*> data;
```

Part B.
Fill the vector from part A (above) with addresses of 26 blocks that you allocate on the heap. Each Block will have its label attribute hold a single uppercase letter from the English alphabet.

```
int i =0;
while (i < 26){

      Block* newBlock = new Block{label}; //lab transaction

      data.push_back(newBlock);

      i++;

      label++;

}
```

Part C.
Modify the labels by adding the integers 32 to the label that was stored in the Block. For example, after modification, the Block that stored 'A' as its label will be modified to store 'a' as its new label.

```
for(size_t i=0; i<data.size();++i){
      data[i] -> label+=32;
}
```

Part D.
Finally, free all of the space you allocated on the heap.Do not leave any dangling pointers.

```
for (size_t i=0, i<data.size(), ++i){
      delete data[i];
      data[i] = nullptr;
}
```

Question 11
For this problem, you will be asked to write two functions:

- ➔ one will read a file that contains data about players in a video game tournament and fills a vector that stores players' information
- ➔ one that will display those players whose scores match or exceed a threshold value

The Input File
Each line of the Input file should be represented as a Player struct. (See the example data file under the section Example of the Input file)

The struct Player is defined as:

```
struct Player{
        int region;
        string handle;
        int score;
};
```

(1) read_data Function

Implement the read-data function (on the next page). Your implementation should accept an input stream and a vector for storing the data. The function should populate the vector and return nothing.

(2) promote function

Implement the promote function (on the next page). Your implementation should accept a vector of Players and an integer value. Print any player that has a score that is greater than or equal to the integer parameter. Output in any format.

read_data Function

```
void read_data(ifstream ifs, vector<Player>& contestants){
      int region;
      string handle;
      int score;
      while(ifs >> region >> handle >> score){
            Player play;
            play.region = region;
            play.handle = handle;
            play.score = score;
            contestants.push_back(play);
      }
}
```

Promote Function

```
void promote(vector<Player> players, int score){
      for (int i =0; i< players.size(); i++){
            if(players[i].score >= score){
                  cout<<players[i].region<<" " << players[i].handle<<" " <<
            players[i].score << endl;
            }
      }
}
```

Given the following definition of a Pixel struct, complete the 3 parts

```
struct Pixel{
        int red;
        int blue;
        int green;
};
```

Part A.

Define a function that accepts ifstream and vector of Pixel pointers. The input file contains 3 integers per line with each integer corresponding to the red, blue, and green RGB colors (in that order) for a pixel. An example file could contain the following lines:

0 23 56
9 245 3

The function fills the vector with pointers to Pixel objects, one for each line in the file.

```
void fill(ifstream& ifs, vector<Pixel>& pixels{
        while(ifs >> red>>blue>>green){
                pixels[i]-> push_back(red);
                pixels[i]-> push_back(blue);
                pixels[i]->push_back(green);
        }
}
```

Part B

Define a function that accepts a vector of Pixel pointers. For every Pixel with a pointer stored in the vector, calculate the grayscale value for the pixel and output it to a separate line of standard output. The grayscale value for a pixel is calculated as the average of the Pixel's red, blue, and green values.

```
void greyScale(const vector<Pixel*>& pixels){
	for (size_t i=0; i=pixels->size(); ++i){
		int redVal =pixels[i]->red;
		int blueVal =pixels[i]->blue;
		int greenVal =pixels[i]->green;
		int grey = (redVal + blueVal+greenVal)/3;
		cout<<grey<<endl;
	}
}
```

Part C

Define a function that frees the memory allocated on the heap for each item in a vector of Pixel pointers. Do not leave any dangling pointers.

```
void clear(Pixel** pix){
	for(size_t i = 0;i<10; ++i){
		delete pix[i];
		pix[ i ] = nullptr;
	}
	// if it was an array, also delete array
}
```

Write a function readAccounts that is passed in input file stream and a vector of accounts. The input file stream is already open you do not have to check. Your function will fill the vector from the information in the file.

Part A

Write a function displayAccounts that is a vector of Accounts and prints out for each account the name of each account and the total of the withdrawals for the account.

Part B

```
void displayAccounts(const vector<Accounts>& accounts){
for(size_t i =0; i< accounts.size(), ++i){
cout<< accounts[i].name << “”<< withdrawals.size()<<endl;
}
```

Long Answer

Question 12

An attire object is defined as follows:

- it contains a description
- a color
- a collection of Attire objects that can be matched with this attire object.

Implement the following functions

- An appropriate constructor for the Attire class
- Overload the output operator for an Attire object providing the information about the Attire object. This must include the description, color and the color/description of the Attire object(s) that can be matched with this Attire object,if applicable
- a method matches_with which adds an attire object ot the vector of Attire objects that match

Enforce the following rules

- to keep this problem relatively simple, an Attire object matches with another Attire object when they share the same color
- An attire object can not match with itself
- An attire object cannot be added more than once to the “matches” collection for another Attire object (but can match with more than one Attire object)

```
class Attire{
	friend ostream& operator<<(ostream& os, const Attire& rhs);
	private:
		string description
		string color;
		vector<Attire*> matches;
	pubic:
		Attire(const string& description, const string&
		color):description(description), color(color){}

		bool matches_with(Attire& attire){
		    if(this == &attire)
		        return false;
		    for(size_t i=0;i<collection.size();i++)
		    {
		        if(collection[i] == &attire)
		            return false;
		    }
		    if(color == attire.color)
		    {
		        collection.push_back(&attire);
		        attire.collection.push_back(this);
		        return true;
		    }
		    return false;
		}
	return false;
}
ostream& operator<<(ostream& os, const Attire& rhs){
	os<<rhs.description<< " ( " << rhs.color<< ") ";
	if (rhs.matches.size()==0){
		os<< " " << endl;
	}
	else{
		os<< "matches with";
		for(size_t i=0; i<rhs.matches.size(); ++i){
			os<<rhs.matches[i]-> color << " " <<rhs.matches[i]->
		description <<" ";
		}
		os << endl;
}
return os;
```

Provide the definition of the Tool class.
Implement the following functions

- an appropriate constructor for the tool class (see the test code below)
- Overload the output operator for a tool providing the informatio about the Tool. This must include the name, voltage and the same names of the tools that can be used with this tool, if applicable
- a method works_with which adds a tool to the vector of tools that supports the use of this tool

```
class Tool{
	friend ostream& operator<<(ostream &out, const Tool&t)
private:
	string name;
	int volts;
	vector<Tool*> tools;
public:
	Tool(string name, int volts)
	{
		this -> name = name;
		this-> volts = volts;
	}

	bool works_with(Tool& other)
	{
		if(volts == other.volts)
			return false;
		tools.push_back(&other);
	return true;
	}
ostream& operator<<(ostream &out, const Tool&t)
	{
		os<<t.name<< ": (" << t.volts << ")";
		if(t.tools.size() != 0)
		{
		    cout<< "works with: ";
```

1.3 Assignment Operators

Implement an appropriate assignment operator

```
class WeatherReading{
public:
	WeatherReading(string s): s(s) {}
private:
	string s;
};

class Image{
public
	Image(string s):name(s), p(new WeatherReading(s)){}
	~Image() {delete p;}
private:
	WeatherReading* p;
	string namel
};
```

Given the following definition of the CellPhones class,

```
class CellPhone{
public:
        CellPhone (double size =0):size(size) {bt = new battery;}
        ~CellPhone() {delete bt;)

private:
        double size;
        Batter* bt;
};
```

Implement the assignment operator for the CellPhone class such that it can be invoked by the assignment statement below.

```
cellPhone& operator = (const CellPhone& cell){
      if (this != &cell) {
            delete bt;
            size = cell.size;
            bt = new Battery
            *bt = *cell.bt;
      }
      return * this;
}
```

FA 2017

```
class ShuttleCraft{
public:
	ShuttleCraft(string s):s(s) {}
private:
	string s;
};

class Orville{
public:
	Orville(string s):p(new ShuttleCraft(s)) {}
	~Orville() {delete p;}
private:
	ShuttleCraft* p;
};
```

Implement an appropriate assignment operator, i.e. a deep copy, for the class Orville. Write it below.

```
Orville& operator= (const Orville& rhs){
	if (this != &rhs) {
		delete p;
		*p = *rhs.p;
	}
	return * this;
}
```

```
class Dragon{
public:
	Dragon(string s):s(s) {}
private:
	string s;
};

class Falcon
public:
	Falcon(string s):name(s), p(new Dragon(s)) {}
	~Falcon() { delete p;}
private:
	Dragon* p;
	string name;
};
```

Implement an appropriate assignment operator, i.e. a deep copy, for the class Falcon. Write it below. And yes, the class Dragon supports copy control.

```
Falcon(const Falcon& other){
	p=new Dragon(*other.p);
	name=other.name;
}
```

Consider the existence of a cass named Point. Two point objects have been instantiated by the statements below:

Point pt1(3.2, 4.6);
Point pt2(9.1, -4.2);

A third Point object is instantiated by the statement below.

Point pt3 = pt1;

The statement above produces which of the following function calls?