|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **National University of Computer and Emerging Sciences, Lahore Campus** | | | | |
| C:\Users\saif\AppData\Local\Microsoft\Windows\Temporary Internet Files\Content.Word\final design.jpg | **Course:** | **Object Oriented Programming** | **Course Code:** | **CS 217** |
| **Program:** | **BS(Computer Science)** | **Semester:** | **Spring 2019** |
| **Duration:** | **60 Minutes** | **Total Marks:** | **20** |
| **Paper Date:** | **12-Apr-2019** | **Page(s):** | **5** |
| **Section:** | **ALL** | **Section:** |  |
| **Exam:** | **Midterm Exam 2** | **Roll No:** |  |
| **Instruction/Notes:** | 1. Answer in the space provided 2. You can ask for rough sheets but **they will not be graded or marked** 3. In case of any ambiguity, make a reasonable assumption. Questions are not allowed | | | |

**Question 1:**  **(Marks: 5)**

Considering class A, B and C write the output of main function.

Note that there is no error in the code, therefore, you do not need to make any modification in the code.

|  |  |
| --- | --- |
| class A{  int a;  public:  A():a(0) {  cout<<"A()"<<endl;  }  A(int a):a(a){  cout<<"A(int a)"<<endl;  }  ~A(){  cout<<"~A()"<<endl;  }  void print() {  cout<<"a="<<a<<endl;  }  void seta(int a){  this->a=a;  cout<<"seta(int a)"<<endl;  }  protected:  void prot\_func\_A(){  cout<<"prot\_func\_A()from A"<<endl;  }  }; | class B: protected A{  int b;  public:  B():b(0) {  cout<<"B()";  }  B(int b, int a=0):b(b) {  cout<<"B(intb, inta=0)"<<endl;  seta(a);  }  ~B() {  cout<<"~B()"<<endl;  }  void print(){  cout<<"b="<<b<<endl;  A::print();  }  void prot\_func\_A(){  cout<<"prot\_func\_A() fromB"<<endl;  }  }; |
| void main() {  B objB(10,20);  objB.print();  objB.prot\_func\_A();  } | **OUTPUT:** |

**Question 2:**  **(Marks: 15)**

Consider a game in which players have some special powers, which they can use to attack opponent players, and to defend from an attack. Here are defined the C++ classes **Player** and **Power** to implement this game.

class Power{

char \* name;

int points;

public:

Power();

~Power();

Power(char\* name, int points);

Power(Power & p);

char \* getname();

int getPoints();

friend ostream & operator<<(ostream &, const Power &);

};

class Player{

int playerId;

Power \*\* powers;

int maxPowers;

int currPowers;

public:

Player(){

playerId = 0;

maxPowers = 0;

currPowers = 0;

powers = nullptr;

}

};

Consider the following driver program. Add all necessary methods, with complete implementation, to the **Player** class, so that driver program works properly, without any compile, run time or logical errors. Specifically: provide the necessary constructors and operators. Read the comments in the driver program to get a hint about how the methods work.

You can assume that all listed functions in class **Power** are already implemented.

void main(){

Power p1("Fire", 20);

Power p2("Water", 30);

Power p3("Wind", 15);

Power p4("Earth", 25);

Power p5("Freeze", 40);

Power p6("Super Speed", 30);

Player Captain(1000, 4); //create Player with maximum of 4 Powers.

Player Planet(1200, 5); //create Player with maximum of 5 Powers.

Captain + p1; //Add one power in the Captain (Deep Copy)

Captain + p3 + p2; //Add two powers in the Captain (Deep Copy)

Planet + p3 + p4 + p5; //Add 3 powers in Planet (Deep Copy)

//it should avoid addition of same power again

Captain - "Wind"; //Searches and removes the power of wind from Captain's Power

//Double the player Ids when copy one player to another one.

//Add powers of Captain and Planet,

//it should also avoid addition of same power again

Player Hero = Captain + Planet + p6;

//Prints all Players complete information

cout << Captain << Planet << Hero;

}