2014-15 | A

1(a)

C	C++
~~cout, cin~~ printf and scanf for formatted I/O	printf, scanf are the least used, might cause problems if mixed with cin, cout, cin, cout are used
Inclusion of header files #include<math.h>	Inclusion of headers #include<cmath> (.h is not used as header)
No use of stream manipulators, rather formatted output is used	Usage of flags & manipulator functions
Inclusion of stdio.h	Inclusion iostream
Namespaces ~~no~~ are not used	Usage of namespaces (eg. std ~~::ios~~ etc.) to avoid name collisions
for each loops are not used	for each loops & usage of iterators

1(b) → Out of Syllabus

1(c) Problems:

① *p = x MUST be inside the constructor, writing it outside of the scope will cause undeclared

variable ~~error~~ error.

(ii) Missing ; after class declaration

(iii) Usage of dynamic memory allocation needs copy constructor, otherwise ~~do~~ multiple freeing up of the same pointer (due to bitwise address copying) or unintended changes might occur, ∄ a Copy constructor.

(iii) Defining destructor inside constructor

(iv) C++ semantics doesn't allow definition of function within function, so it's a serious breach of syntax, causes compilation error. It must be a standalone function which will be invoked automatically.

The copy constructor:

```
myclass (const myclass& obj){
    this->p=(int* )malloc(sizeof(int));
    if(p==nullptr)
    {
        cout<<"Copy unsuccessful due to memory
              allocation problem\n";
        exit(1);
    }
```

```
*p = *obj.p;
}
```

Working procedure:

① First for our objects member *p, size of an int is allocated in the heap.

② For safety, allocation success is checked

③ Then only the value is copied to the object.

2(a)
```
int& f(~~int x~~)
{ return x;
}
```

we can do the following;

```
f() = 5;
```

which is same as x = 5;

Thus a fnc. call can be written on the left side of = operator.

2(b)
```
int main()
{
  void (*p1)(char) = &space;
  void (*p2)(char, int) = &space;
  p1('a');
  p2('a', 5);
}
```

2(c)
```
class complex{
    double ~~real~~ x, y;
    complex(double x, double y)
    {
        this->x = x;
        this->y = y;
    }
complex operator+(complex obj)
{
    complex temp;
    temp.x = x + obj.x;
    temp.y = y + obj.y;
    return temp;
}
complex operator++()
{
    ++(this->x);
    ++(this->y);
    return *this;
}
complex operator++(int unused)
{
    complex temp ~~= *this~~;
    temp.x = this->x;
    temp.y = this->y;
    ++(this->x);
    ++(this->y);   return temp;
}
```

```
complex operator+(double num){
  complex temp;
  temp.x = x + num;
  temp.y = y + num;
  return temp;
}
```

~~complex~~ friend complex operator+(double num){

```
    complex temp;
    temp.x = x + num;
    temp.y = y + num;
    return temp;
}
```

3 (a), (b) → Kindly see slides

(c) No modification is needed except for the addition of two ~~member~~ friend functions

```
friend ostream& operator<<(ostream& out, inventory inv)
{ out<<"item: "<<item<<"onhand: "<<onhand
        <<"cost: "<<cost<<endl;
  return out;
}
friend ~~ostream friend~~ istream& operator>>(istream& in, inventory& inv)
{
  in>>~~ch~~ item;
  return in;
}
```

5(d)

```
ostream& setup(ostream& out){
    out<<setw(10)<<left<<setfill('%');
    return out;
}
int main()
{
    cout<<setup<<123.45678;
}
```

→ Also see 14_15_4a.cpp in GitHub.

4(a)

```
class stdId{
public:
    string ID;
};
```

```
#include<iostream>
#include<map>
using namespace std;
class student{
public:
    string name;
    string contact;
    friend ostream& operator<<(ostream& out, student stdnt)
    {
        out<<stdnt.name<<" "<<stdnt.contact;
        return out;
    }
    // this part is not necessary (↓)
    friend istream& operator>>(istream& in, student& stdnt)
    {
        in>>stdnt.name>>stdnt.contact;
        return in;
    }
```

```
student(string name, string contact){
    this->name=name;
    this->contact=contact;}
};
class stdId {
public:
    string ID;
    stdId(string ID){
        this->ID=ID;
    }
    friend ostream& operator<<(ostream& out, stdId id);
    friend istream& operator>>(istream& in, stdId& id);
};
ostream& operator<<(ostream& out, stdId id)
{
    out<<"ID: "<<id.ID;
    return out;
}
istream& operator>>(istream& in, stdId& id)
{
    in>>id.ID;
    return in;
}
class customCmp {
public:
    bool operator()(const stdId& id1, const stdId& id2) const
    {
        return id1.ID<id2.ID;
    }
};
int main(){
    map<stdId, student, customCmp> ID_stdnt;
```

```
student s1("Ibtida", "012345");
student s2("Sakif", "012346");
student s3("Mr. Hello World", "01456");
student s4("Null Pointer Brother", "012345");
stdId id1("2105061");
stdId id2("2105065");
stdId id3("2105183");
stdId id4("2105184");
ID_stdnt.insert(make_pair(id1, s1));
ID_stdnt.insert(make_pair(id2, s2));
ID_stdnt.insert(make_pair(id3, s3));
ID_stdnt.insert(make_pair(id4, s4));
cout << "Enter student ID: ";
stdId temp(" ");
cin >> temp;
auto it = ID_stdnt.find(temp);
if (it == ID_stdnt.end())
{
    cout << "Not found\n";
}
else cout << it.second << endl;
}
```

4(b)
```
#define SIZE
template <class t>
class queue
{
   t* arr;
   int head, tail;   int size;
 public:
   queue(int size)
   { arr = new t[size+1];
     head = 0;
     tail = 1;     this -> size = size;
   }
  ~queue() {
     delete[]arr;
   }
 void enqueue(t item){
   if((tail+2)%(size+1) == head)
   { cout<<"Queue is full!!";
     return;
   }
   arr[(++
   tail = (tail+1)%(size+1);
   arr[tail] = item;
 }
```

```
t dequeue(){
  if(~~rear~~ tail - front == 1 || front - tail == 1)
    {cout<<"Queue is ~~full~~ empty"<<endl;
     return -1;
    }
  t ret = arr[head];
  head = (++head)%(size+1);
  return ret;
}
t front(){
  ~~ret~~ if(tail - front == 1 || tail - front == -1)
    {cout<<"Queue is empty"<<endl;
     return -1;
    }
  return arr[head];
}
bool isEmpty(){ return(tail - front == 1 || tail - front == -1);}
bool isFull(){ return !(abs(tail - front) == 1);}
int ~~size~~ currentSize(){
  return (~~head~~ = tail - head + size)%(size+1);
}
};
```

Ref:
4(c) → stack over flow

B

5(a) (i) int c[], x; means c is an array of integers (primitive) and x is an int variable

(ii) int[] c, x; means both c and x are integer type arrays.

Two ways:

(i) int a[][] = {{0} {0,0}, {0,0,0}, {0,0,0,0}, {0,0,0}, {0,0}, {0}};

(ii) int a[][] = new int[7][];

```
a[0] = new int[1];
a[1] = new int[2];
a[2] = new int[3];
a[3] = new int[4];
a[4] = new int[3];
a[5] = new int[2];
a[6] = new int[1];
for(int i=0; i<b.length; i++)
{
  for(int j=0; j<b[i].length; j++)
    {
      Syst b[i][i] = 0;
    }
}
```

5(b)

```
public class TestMain{
static int testf(String optype, int ... numbers)
{
  int ans = 0;
  if(optype.equals("sum"))
  {
    for(int num:numbers)
      ans += num;
  }
  if(optype.equals("mult"))
  {
    ans = 1;
    for(int num : numbers)
      ans *= num;
  }
return ans;
}
public static void main(String[] args){
  //Rest
  }
}
```

5(c)

~~Integer a = 5;~~

```
int a = 1234;
```

String s1 = Integer.toString(ab); //way#1
String s2 = String.valueOf(ab); //way#2

//Code:

```
public class TF{
  public static void main(String[] args)
  { Integer max_min[] = { Integer.MIN_VALUE, Integer.
                          MAX_VALUE};
    for(int i=0; i<args.length; i++)
    { Integer x = Integer.parseInt(args[i]);
      max_min[0] = Integer.max(max_min[0], x);
      max_min[1] = Integer.min(max_min[1], x);
    }
    System.out.println("max: " + max_min[0] +
                       " min: " + max_min[1]);
  }
```

CMDs: javac TF.java ↵

java TF 12 -5 400 10 100 150

400 -5

---

5 (d)

```
MyClass{
  static int count=0;
  MyClass(){
    ++count;
    if(count>100)
      count=0;
  }
} //(Ans)
```

6(a) github.com/Ibtida01/TF_Codes_1_2/blob/main/20_21_6a.cpp

6(b) Replacing //your code with the following code:

```
void f2(){
    //Body
}
void f3(){
}
void f4(){
}
void f5(){
}
```

6(c)

```
interface i1{
    void f1(); default void f1(){//...}
    void f2(); default void f2(){//...}
}
interface i2{
default void f3(){ //Body }
default void f4(){ //Body }
}
interface i3{
  default void f5(){ //Body }
```

```
    default void f6() { //Body }
}
interface i4 {
    void f7();
}
MyClass implements i4 {
    void f7() {
        //Body
    }
}
```

6(d) 3 uses: (i) a class having final keyword in its signature, can't be extended by any other class

(ii) a method declared as final can't be overriden further

(iii) final variables act as constants, changing their values will result in compile time error.

command:

```
javac -d . Balance.java
java MyPackage.Balance
```

7(a) github.com/Ibtida01/TF_Codes1_2/tree/main

(b) → Producer Consumer Problem, out of syllabus

(c) Code Link: t.ly/voo2s → (or follow Link in 7(a))

(d) → github.com/IbtidaC1/TF_Codes_1_2/tree/main

8 (a) → (b), (c), (d) → [Same]