1)

Exception handling

One of the advantages of C++ over C is Exception Handling. Exceptions are run-time anomalies or abnormal conditions that a program encounters during its execution. There are two types of exceptions: a)Synchronous, b)Asynchronous

C++ provides following specialized keywords for this purpose.
- try: represents a block of code that can throw an exception.
- catch: represents a block of code that is executed when a particular exception is thrown.
- throw: Used to throw an exception. Also used to list the exceptions that a function throws, but doesnât handle itself.

Following are main advantages of exception handling over traditional error handling.

1) Separation of Error Handling code from Normal Code: In traditional error handling codes, there are always if else conditions to handle errors. These conditions and the code to handle errors get mixed up with the normal flow. This makes the code less readable and maintainable. With try catch blocks, the code for error handling becomes separate from the normal flow.

2) Functions/Methods can handle any exceptions they choose: A function can throw many exceptions, but may choose to handle some of them. The other exceptions which are thrown, but not caught can be handled by caller. If the caller chooses not to catch them, then the exceptions are handled by caller of the caller.
In C++, a function can specify the exceptions that it throws using the throw keyword. The caller of this function must handle the exception in some way (either by specifying it again or catching it)

3) Grouping of Error Types: In C++, both basic types and objects can be thrown as exception. We can create a hierarchy of exception objects, group exceptions in namespaces or classes, categorize them according to types.

Example Program:

```
#include<iostream>
using namespace std;

int main()
{
    int x;
    int y;
```

```
        cin>>x>>y;
        try {
            if(y<0) {
                throw “Divide by zero”;
            }
        }
        catch(string x) {
            cout<<x<<endl;
        }
        catch(...) {
            cout<<"Catch ALL";
        }
    }
```

C++ does not support 'finally' blocks. The reason is that C++ instead supports RAII: "Resource Acquisition Is Initialization"

The idea is that an object's destructor is responsible for freeing resources. When the object has automatic storage duration, the object's destructor will be called when the block in which it was created exits -- even when that block is exited in the presence of an exception.

---

2)

Types of relationships

(a) Inner Class – Composition

(b) Nested Class – Commopsition

(c) Inheritance – IS A reltionship

---

3)

I/O Streams in Java and C

A stream is a sequence of data bytes which is a modular representation of data in real world.It is called stream mainly because it continuous to flow like any fluid such as water.

In Java there are 3 main streams which are automatically created and are linked with the console by default.

System.in - Standard Input Stream

System.out - Standard Output Stream

System.err - Standard Error Stream

In C all devices are treated as files by the language.So, devices such as the display are addressed in the same way as files and the following files are automatically executed when a program is executed to provide access to the keyboard and mouse interfaces.

Standard File - File Pointer Device

Standard Input - stdin Keyboard

Standard Output - stdout Screen

Standard Error - stderr Your Screen

The file pointers are means to access the files, read and write data and information.

---

4)

Java doesn't support Operator Overloading for the following reasons

- Simplicity and cleanliness — Individual programmer overloading an operator may spoil the readability and may also complicate the process of compilation and execution
- JVM complexity
- Avoid programming errors — With definition of operators becoming more complex,there are more chances of errors creeping in the program.
- Easy Development Tools — With overloading development becomes very complicated and hence for the purpose of easy development, overloading is not supported by Java.
