OOP

Operator Overloading

Overloading means to redine or restate somethings way of usage In terms of operators we can say that to define multiple way of usage for one operator.

Overloading 'X' Operator:

→ Relational operators

It returns the bool value as a result, so in order to return bool value its syntax is bool.

```
bool ClassName :: operator>(const Class
    & obj);
```

→ Cout & Cin operators:

Overloading these operators is done in a slightly different way, however, than other operators. These operators are actually part of the ostream and istream classes defined in the C++ runtime library.

⇒ you must write operator functions to overload the ostream version of << the istream version of >>, so they work directly with a class.

ostream &operator << (ostream & strm,
const ClassName & obj)
{ strm << obj.members;
return strm;
}

→ This reference means that it returns a reference not the actual copy of the obj.

Using as friend

Some compilers require you to prototype the >> and << operator functions outside the class. For this reason, we have added the following statements to Feetinches.h class specification

Class Feetinches;
ostream &operator << (ostream & strm, const Feetinches &);
istream &operator >> (istream &, const Feetinches &);

→ [] overloading

index

int &IntArray :: operator [] (const int &);

Object Conversion:

Special operator functions may be written to convert a class object to any other type

Aggregation:

Aggregation occurs when a class contains an instance of another class.

Example:

A house is made of door object window, window object etc all included in a single object house.

Explanation:

Making an instance of one class an attribute of another class. The word aggregate means the whole that is made of constituent parts.

⇒ When an instance of one class is a member of another class, it is said that there is a "has a" relationship b/w them.

⇒ For example: The house has a door.
The house has a window

⇒ It is sometimes called part-whole relationship bcz one object is part of a greater whole.

Inheritance:

Inheritance allows a new class to be ~~built~~ based on existing class. The new class ~~Inherits~~ inherits all the member functions (except the constructors and destructors) of the class it is based on.

⇒ For example: Insects have grasshopers and bees in its class

⇒ grasshopers have all insects properties as well as their special ability to jump.

⇒ Same is for bees they can sting as well.

⇒ we write "is a" in inheritance

⇒ Grasshopper is an insect

⇒ bee is an insect.

⇒ A car is a vehicle

⇒ A rectangle is a shape.

Protected:

Protected are like private members, but they can be accessed by derived class