Some More Concepts in C++

Type Conversion

- 1. Primitive to primitive type:- automatically done by compiler
- 2. Primitive to class type:- It is done by using constructor.

```
for e.g.
```

```
class Complex
private:
    int a,b;
public:
    Complex(){}
                   → As the c1 is created default constructor is invoked
    Complex(int k){
    a = k; b = 0;
    void setData(int x, int y){
    a=x; b=y;
    void showData(){
    {cout<<"\n a:"<<a<"b:"<<b;}
};
void main()
{
    Complex c1;
    int x=5;
    c1 = x; \rightarrow primitive to class – invokes the constructor with x as argument
    c1.showData();
```

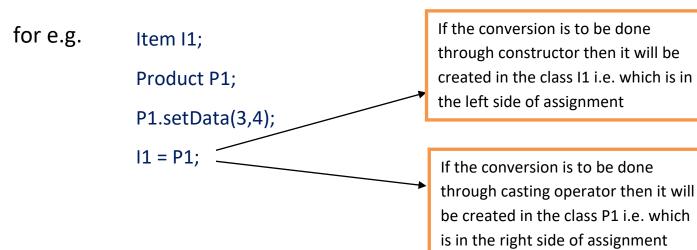
3. <u>Class to primitive type</u>:- It can be implemented with casting operator(inside the class).

```
operator type(){
the primitive type
to which we want
to convert

for e.g.
```

```
class Complex
private:
    int a,b;
public:
    void setData(int x, int y){
           a=x; b=y;}
    void showData(){
          {cout<<"\n a:"<<a<"b:"<<b;}
    Operator int(){
          return a;} → It depends on you what you want to return i.e. a or b.
};
void main()
    Complex c1;
    int x;
    x = c1; \rightarrow class to primitive
    cout<<"\n x ="<<x;
```

- 4. **Class to another class type:-** This type of conversion can be implemented by two ways.
 - i) Conversion through constructor
 - ii) Conversion through casting operator



```
class Product{
    private:
       int m,n;
    public:
       void setData(int x, int y)
          {m=x; n=y;}
       int getM()
          {return m;}
      int getN()
         {return n:} };
class Item(){
private:
      int a,b;
public:
      void showData()
       {cout<<''\n a = "<<a<''b = "<<b:}
      Item(){}
                   →default constructor
      Item(product p){
            a = p.getM();
            b = p.getN(); };
int main(){
  Item I1;
 Product P1;
  P1.setData(3,4);
  I1 = P1;
 I1.showData(); }
```

Exception Handling

Exception is any run time error which is run time error

- Exception are off beat situation in your program where your program should be ready to handle it with appropriate response.
- C++ provides a built-in error handling mechanism that is called exception handling.
- Using exception handling, you can more easily manage and respond to run time errors.

try, catch, throw:-

- → Program statements that you want to monitor for exceptions are contained in a try block.
- \rightarrow If any exception occurs within the try block, it is thrown(using throw).
- → The exception is caught using catch keyword and processed.

Syntax:

```
try{
.....
throw constant}
catch (type 1 arg.){
}
catch (type 2 arg.){
}
....
catch (type N arg.){
}
```

NOTE:-

- 1. throw keyword is used under try block.
- 2. try and catch blocks are always written in pairs, one can't exist without other.
- 3. catch is always written just after try.
- 4. If only throw is written, then the program will run, but it will terminate in the middle of program.
- 5. We can also use a function in the place of throw which will contain throw in body.
- 6. If one catch is executed then subsequent catch blocks will never execute.

for. e.g.

NAMESPACE

- \rightarrow It is a container for identifiers.
- → It puts the name of its members in a distinct space so that they don't conflict with the names in other namespace or global namespace.
 - Q. How to create namespace

- → Namespace definition does not terminate with a semi colon like in class definition.
- → The namespace definition must be done at global scope, or nested inside another namespace.
- → You can use an alias name for your namespace for ease of use like wise namespace ms = Myspace;
- → Namespace is not a class, you cannot create instance of namespace.
- → A namespace definition can be continued and extended over multiple files, they are not redefined or overridden.

```
file1.h

namespace Ms
{

int a,b;

void f1();
}
```

```
file2.h

namespace M1

{
    int x,y;
    void f2();
}
```

Accessing members of namespace:-

Any name(identifier) declared in a namespace can be explicitly specified using the namespace's name and the scope resolution :: operator.

```
e.g.
#include<iostream>
using namespace std;
namespace Myspace
    int a;
    void f1();
    class hello1{
      public:
          void hello1()
          {cout<<"Hello":}
    };
 void Myspace :: f1()
 { cout << "In f1";}
int main(){
    Myspace :: a = 5;
    Myspace :: Hello obj;
    obj.hello1();
    Myspace :: f1();
```

<u>Using</u>:- Using is a keyword which allows to import an entire namespace into your program with a global scope. It can also be used to import a namespace into another namespace or any program.

```
file1.h

namespace Myspace
{
    int a,b;
    class A{;
};}
```