CH-3: OPERATORS AND TYPE CASTING:-

Operators:- It is a symbol used for programming performing operations on operands. These operations can be mathematical or logical.

There are diff. types of operators in C++ for performing diff. operations.

i) **Assignment operator:-** Assign values to variables. The operand / variable is added to left side of the operator while the value added to the right side of the operator.

The variable & the value must belong to the same d-type, otherwise, the C++ compiler will raise error.

e.g:- `int x = 50;`

`=; +=; -=; *=; /=`

Used to perform arithmetic / mathematical operations.

```
#include<iostream>
using namespace std;
int main()
{
 int a,b, sum;
 cout<<"Enter two numbers";
 cin>>a>>b;
 sum=a+b;
 cout<<"Sum of entered number is="<<sum;
 return 0;
}
```

ii) **Relational operators:-** perform comparisons on operands

`==; != ; >; <; >=; <=`

e.g: `if(a==b)`

iii) **Logical operators:-** Used to determining the logic b/w variables or values.

`&& ; || ; !`

iv) **Bitwise Operators:-** works on bits & performs bit-by-bit operation

`&; |; ^; ~; << ; >>`

p	q	p&q	p|q	p^q
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

```
#include<iostream>
using namespace std;
int main()
{
 int a=10, b=7;
 cout<<(a&b);
}
```

v) Unary operator :- It funcn to produce a new value on a single operand. (&X Address-of) ; (+) ; (-) ; (++)-Prefix increment operator ; (--) ; sizeof ; Cast operator ; () - explict conversion of the type of an obj in a specifica situation.

```
int main()
{
  int x;
  float y = 48.23;
  x = (int) y;
  x = -x;
  cout << x;
  return 0;
}
```
⇒ -48

vi) Ternary / Conditonal Operator : evaluates the test condition & excutes a block of code based on the result of the condition.

Syntax:-
condition ? expression1 : expression2;

vii) Misc operator :-

operator	operand	operation	details
sizeof	a	sizeof(a)	It returns the memory occupied by the particular d-type
&	a	&a	It refers to the address
*	a	*a	is a pointer
?:	a,b	a?b; Statement	An alternative for if-else condition

#Scope Resolution operator :-

Used to reference the global variable or member funcn that is out of scope. It is represented as a double colon (::) symbon

Uses:-

- Used to access the hidden variables on member funcn of a program.
- Defines the member funcn outside of the class using the scope resolution.
- Used to access the static variable & static funcn of a class.
- The scope resolution operator is used to override funcn in the inheritance.

```
#include<iostream>
using namespace std;
int num = 100;
int main()
{
  int num = 750;
  cout << "The value of the local variable num: " << num << endl;
  cout << "The value of the global variable num: " << ::num;
  return 0;
}
```

```
#include <iostream>
using namespace std;
class demo
{
public:
    void show();
};
void demo::show()
{
  cout<<"show is member function of class demo";
}
int main()
{
  demo d;
  d.show();
  return 0;
}
```

show is member function of class demo

Member dereferencing operator:-

- The pointer-related operator & and * are called referencing & dereferencing operators.
- Referencing operator (&) is a unary operator & it returns the address of its operand variable.
- The dereferencing operator (*) is a unary operator that returns the value present at the specified address.
- Using a pointer we can access the member of the class

Pointers to member operator:-

.* - To access a member using object name

->* - Access a member using a pointer to the object.

```
#include <iostream>
using namespace std;
class demo
{
  public:
    int m;
    void display()
    {
      cout<<"x = "<<m<<endl;
    }
};
int main()
{
  demo A;
  demo * ptr;
  int demo::* k=&demo::m;
  A.m=10;
  ptr=&A;
  cout<<ptr->*k;
  ptr->display();
  return 0;
}
```

⇒ 10 x = 10

Reference Vs. Pointers :-

- Once a reference is created, it cannot be later made to reference another object.
- Reference cannot be NULL. Pointers can be NULL
- A reference must be initialized when declared. There is no such restriction with pointers.

Type Conversion in C++ :-

C++ allows to convert one dtype to another dtype.

Types:-

1) Implict type conversion:- Done by the compiler on its own, without any external trigger from the user.
Generally takes place when in an expression more than one dtype is present. In such condition type conversion takes place to avoid loss of data.

```
#include<iostream>
using namespace std;
int main()
{
   int x=100;
   char y='a';
    x=x+y;
   float z=x+1.0;
  cout<<"x="<<x<<endl
      <<"y="<<y<<endl
      <<"z="<<z<<endl;
  return 0;
}
```

2) Explicit type conversion :- This process is user defined, also called type casting. This process is complete converting by assignment.
Syntax:- (type) expression

```
#include<iostream>
using namespace std;
int main()
{
   double x=25.5;
   int sum=(int)x+1;
   cout<<"SUM="<<SUM;
   return 0;
}
```

Possible type Conversion in C++ :-

- **Conversion from basic type to class type :-**

Source type is basic type & the destination type is class type.
Basic d-type is converted into class type.

Possible ways to perform basic type conversion :-

1. Using Constructor :-

```
#include<iostream>
using namespace std;
class Time
{
  int hrs, min;
  public:
   Time(int t)
  {
   hrs = t/60;
   min = t % 60;
  }
  void show();
};
void Time::show()
{
  cout<<hrs<<"Hours(s)"<<endl;
  cout<<min<<"Minutes"<<endl;
}
int main()
{
  int duration;
  cout<<"Basic Type to Class Type Conversion\n";
  cout<<"Enter time duration in minutes";
  cin>>duration;
  Time t1(duration);
  t1.show();
  return 0;
}
```

⇒ 14 Hours(s) 10 Minutes

Enter time duration in minutes 850
Basic Type to class Type Conversion
14 : Hours(s)
10 Minutes

- **Class type to basic type :-**

Source type is class type & the destination type is basic type.
Class data type is converted into basic type.
Class type to basic type conversion requires special casting operator func^n for class type to basic type conversion. This is known as the conversion func^n.

Syntax :-

```
operator type_name1()
{
  //code
}
```

Conversion func^n :- It must be a class member.
" " not specify the return value even though it returns the value.
It must not have any argument

```
#include<iostream>
using namespace std;
class Time
{
  int h,m;
  public:
  Time(int a, int b)
  {
    h=a;
    m=b;
  }
  operator int()
  {
    cout<<"\n Class Type to Basic Type
    Conversion...";
    return(h*60+m);
  }
  ~Time() //destructor
  {
   cout<<"\n Destructor called..."<<endl;
  }
};
int main()
{
 int h,m,duration;
 cout<<"\n Enter Hours";
 cin>>h;
 cout>>"\n Enter Minutes";
 cin>>m;
  Time t(h,m);
  duration=t;
 cout<<"\n Total minutes are "<<
 duration;
 cout<<"\n 2Nd method operator
 overloading";
 duration = t.operator int();
 cout<<"\n Total minutes are "<<
 duration;
 return 0;
}
```

⇒ Enter Hours 850
Enter Minutes 1200
Class Type to Basic Type Conversion...
Total Minutes are 52200
2Nd Method operator overloading
Total minutes are 52200
destructor called...

- **One Class to Another Class Type :-**

Both the types that is source & destination type are class type.
Ways :- Using Constructor
Using type conversion func^n.

```
#include<iostream>
using nampespace std;
class Time
{
  int h, m;
  public:
   Time(int a, int b)
   {
    h=a;
    m=b;
   }
   Time()
   {
    cout<<"\n Time's Object created";
   }
   int get Minutes()
   {
   int tot_min=(h*60)+min;
   return tot_min;
   }
  void display()
  {
  cout<<"Hours:"<<h<<"\n";
   cout<<"Minutes:"<<m<<"\n";
  }
};
class Minute
 {
  int min;
  Minute()
   {
    m=0;
   }
   void operator=(Time T)
   {
    m = T.get Minutes();
   }
   void display()
   {
    cout<<"\n Time Total Minutes:"<<m<<"\n";
   }
};
int main()
{
  Time t1(1,20);
   t1.display();
   Minute m1;
   m1.display();
   m1=t1;
   t1.display();
   m1.display(); return 0;
}
```

Using Constructor

⇒ Hours : 1
Minutes : 20
Total minutes : 0
Hours : 1
Minutes : 20
Total Minutes : 80

```
#include <iostream>
using namespace std;
class inventory1
{
  int ino, qty;
  float rate;
  public:
  inventory1(int n, int q, float r)
  {
    ino = n;
    qty = q;
    rate = r;
  }
  int getino()
  {
    return(ino);
  }
  float getamt()
  {
    return(qty * rate);
  }
  void display()
  {
    cout << "\n ino = " << ino << "qty=" 
    << "rate = " << rate;
  }
};
```

Using type conversion function

```
class inventory2
{
  int ino;
  float amount;
  public:
    void operator = (inventory1 i1)
    {
      ino = i1.getino();
      amount = i1.getamt();
    }
    void display()
    {
      cout << "\n ino = " << ino << " amount = "
      << amount;
    }
};
int main()
{
  inventory1 i1(101, 50, 45);
  inventory2 i2;
  i2 = i1;
  i1.display();
  i2.display();
}
```

⇒ ino = 101 qty = 50 rate = 45
ino = 101 amount = 2250