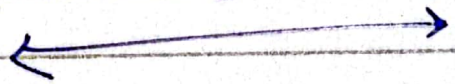
```
NamedSet
{
  String name = "A";
  Set S;

  NamedSet(String n, int cap=9) : Set(cap),
  {
    name = n;
  }

  NamedSet CalculateUnion(const NamedSet& s)
  {
  NamedSet U{ data.CalculateUnion(s)};
  }      return U; int i=0 X
        while( i< U.size)
        d.insert(U.data[i]
```



```
  NamedSet(const setSr) : data(r)
  {

  }

  int Sr = *(new int);
     delete Sr;
```

it is a const pointer

```
A
{
  A &r;
  public:
    A() : r{new A(12)}
    {

    }

    A(int) ; یہاں آ کے گا
    { r تو initialize نہیں
    }
```

r const pointer hai ← initialize نہیں ہوا۔

Nameless Object

String {}; ← nameless → stack
ابھی بنا اور ابھی ختم ہو گیا گا — پھر بنا

String s[2] = {String{}, string{"Hello"}};

int *p = new int [5] {1, 2, 3, 4, 5}; ✓

String *p = new string [5] { nameless objects };

delete p;

↑ Array of objects

Operator Overloading :-

→ Type Cast operator

کون سے operators ایسے ہیں جن کے meaning کو classes کے لیے define نہیں کر سکتے ہیں ۔

(→) , (*.) , (::) , (.) ← x ← (?:) , :: , (*.)

(3)

a = a+b ; →

a += b ;

→ conversion constructor

String a + 12

String a(), b(" ")

a+b ; ⇒ a.operator + (b)

a.concatenate (b)

```
class String {
Public:
  String operator + (String x);
  {

  }
  void operator += (String x) ;
```

Binary operator → Only one parameter

→ a+b

→ c = a+b

→ a += b

→ a = a+b

One argument constructors are called conversion / type cast constructor.

String x ;

Date b,

x + b → b+x

Function should be int String and recieves object type date. | الٹا نہیں

Date {
Date(String x)
{

}

Date operator + (Date);

b + x
↓ ↓
S D

↓
Type cast ہوگی
کرتا ہے / receive String const & Date کیونکہ
ructor

⟵⟶

String x("Hello");
String y("Bye");
x = y
.) ہے گا → leakage Memory
'/x میں Bye آئے گا

↓

String&
↓
x = y
String & operator = (const String & ref)
{
this -> ~String();
if (!ref.data) ← size = ref.size;
return *this; data = new char[size+1];
String copy(data, ref.data);
return * this;
}

(a = b) = c

↳ means ⇒ a = c

a = a → problem

if (&ref == this)

return *this;

String (const String &ref) : String()

{

*this = ref;

}

→ copy constructor

→ assignment function

Index operand;

Array a{5,1,2,5};

a.getset(1) = 30;

a[1] = 30

↳

a.operator[](1)

String x{"hi"}

x + "Hello" →

Scanned with CamScanner

bool operator!() const

{

if (! data || data[0] == '\0')

return true;

return false;

}

→ if (!a)

→ if (a.isEmpty())

int x = !a;

int operator == (String x)

{

-1, 0, 1

}

int operator > (String x)

{

}

Also <

Time t1, t2, t3;

~~t++t1;~~ ++t1;

t1++;

++t1 + t2;

t1++ + t2;

Scanned with CamScanner

++t1; → Pre-increment

t1.operator++()

Time & Time::operator ++ ()

{
  incSec();
  return *this;
}

t1++;

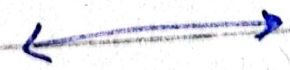
Also → t1.operator++(); → Post-increment

Time Time & operator++(int)

{
  Time a x = *this;
  incSec();
  return x;
}

same for pre and post decrement

.operator SS ( )



Matrix x(3,4)

x[1][2];

int *Matrix :: operator [] (int index)

{
  return data[index];
}

```
a[1][2]

a(1,2) = 30;   → function call operator
               → functor object
```

parenthesis | function call operator

```
int& operator () (int r, int c)
{
    Bound check. (if r> c>)
    return data [r] [c];
}
```

```
string f()
{
    string x, y;
    if (rand()%2==0)   → یہاں پہ compiler کو
        return x;        پتہ نہیں کیا return کرنا ہے
    return y;
}

main ()
    String z = f();
    z = f();
```

یہاں تو مال ہے دوسرا نہیں رکھا جائے گا بلکہ assignment/copy ہو گی

constructor case میں ہوگا بس

```
String (String && r)  → move constructor
{
    data = r.data;
    size = r.size;
    r.data = 0;
    r.size = 0;
}
```

const int & z = 23;

→ can recieve tempory as well L-value

int && y = 32;

→ can recieve only Tempory

String (const string &);
String (String &&);

→ String z = String ("Hello");

دیکھے گا اس کو accept کرنے کے لیے کون ہے copy const ructor ہے
اگر پہلے تو
z کا نام کبھی یہی رکھ دے گا
لیکن copy constructor نہیں چلے گا۔

String (string &) → اس میں یہ غلط copy constructor ہے

→ String z = string ("Hello")

Error آئے گا - کیونکہ وہ دیکھے گا اس کو receive کرنے کے لیے کون ہے copy constructor ایسا نہیں

↔

```
void operator = (String && r)
{
    this → ~String();
    data = r.data;
    size = r.size;
    r.data = 0;
    r.size = 0;
}
```

اگر کوئی object latest ~~compose~~ ہو تو move constructor کال ہوگا

object guest ہو تو move constructor کال ہوگا ?

Swap objects:

```
void swap( String &a, String &b)
{
    String temp = move(a);
    a = move(b);
    b = move(temp);
}
```

```
String d("___");
  = "abc" + d;

cout << a
```

دیکھو کے class
- cout → ofstream
- cin → ifstream

For this :- "abc" + d

As global operator in .h file :-

```
operator + (const char *, String x)
{

}
```

In global :-

```
operator + ( int, int )   X
```

error ہے primitive types دونوں
کوئی ایک parameter user defined ہونا چاہیے

اگر دونوں define کے ساتھ operator same global اور class
میں تو کیا ہو گا ؟

When you define an operator overloading function both as a member function & globally, it can lead to ambiguity and compilation error.

control ostream

return Type

ostream & operator << (ostream & r, const String & s)

{

S.display(); / r.

}

Friend function / friend class

friend ostream & operator << (ostream &, const String &)

friend istream & operator >> (instream & in, String & r)

{

r.input;

}

friend class Date;

→ In String class

access

for this

#include "date.h"

↳ In date Class 6 → Class String; forward declaration

specific class

Type cast operator → Type cast operator
Type کا ناکارہ 6

String s = {"162"}

int x = s ↑ Implicit ; کیسے کریں گے

int x = (int) s ; ↘ Explicit

operator int ( ) ;

↑ یہی return Type ہے ... کا ہی نہیں

String && operator int ( ) → float
Date

{

return toInteger ( );

}

Date d = s "12-15-2016";

میرا operator اگر بنا لیا تو اگر جب use

✓ int x = (int) s; یہ کرتے ہے

explicit String && operator int ( )

{

}

String (int)

string (Date)
String (Time)

Date d; ③

s = d; or s = (string) d;

operator ~~st~~ Date class

operator String ()
{
}

Implicit
Explicit

String (Date)
{
}

More than one user defined conversion ~~function~~ from Date to String.

operate = (Date)
{
}

if (s) { }

operator bool ()
{
}