PAGE:
DATE: / /

- **TOKENS in C++ :**

i) **KEYWORDS** : These are the reserved words. Keywords are the reserved identifiers and cannot be used as a name for the program variable or any other user-defined program elements.

ii) **IDENTIFIERS** : these refers to the name of the variables, functions, arrays, pointers, classes.

iii) **DATA TYPES** :
a) User defined Datatypes : structure, union, class, enumeration.
b) Pre defined Datatypes : int, char, float .etc.
c) Derived datatype : Array, Function, Pointer etc.

* Pre defined : Built in Types
i) Primitive a) long b) char c) Int
ii) Void
iii) floating Point a) float b) double

iv) **CONSTANT** : It refers to the fixed value that do not change during the execution of the program.

v) **STRING** : Collection of characters.

vi) **C++ OPERATORS** :
* a) :: Scope Resolution Operator
b) ::* Pointer to member declaration
c) →* Pointer to Member Operator
d) .* Pointer to Member Operator

Malloc () of C = New of C++ } allocation
Calloc () of C = New of C++

PAGE:
DATE: / /

Malloc of C * e) delete Memory release Operator
Calloc of C * f) new Memory allocation Operator
* g) endl line feed operator

(::)
```
int m = 10;
int main ()    using namespace std;
{ int m = 20;
  { int k = m;
    int m = 30;
    cout << "inner block";
    cout << "k = " << k;          = 20
    cout << "M = " << m;          = 30
    cout << " :: M = " << ::m;
  } cout << "outer block";
    cout << "M = " << m;          = 20
    cout << " :: m = " << ::m;    = 10
    return 0
}
```

Syntax : :: variable_name

- This operator prints the value which is globally declared.

= 10

In C programming, the global version of a variable can't be accessed within the inner block. In C++ resolving this problem is done by introducing the operator ::
this operator allow access to the global version of a variable.
Scope Resolution Operator also defines that a method (function) belong to a particular class.

↓

Syntax = return_type class_name :: function_name
{ =
= }

function's

USING MEMBER FUNCTION

Example :

user defined datatype

↓

```
class student
{ int roll_no ;
  char name[10];
public : void read ();
         void show ();
};
void student :: read ()
{ cout << "Enter the detail" ;
  cin >> roll_no >> name;
}
void student :: show ()
{ cout << "Result is :";
  cout << rooll-no << name ;
}
void main ()
{ student s1 ;
  s1. read ();
  s1. show ();
}
```

CLASS (STUDENT)

} MEMBER FUNCTIONS

DEFINITION OF READ FUNCTION

DEFINITION OF SHOW FUNCTION

DEFINITION OF MAIN FUNCTION

Q: Design a class student without using Scope Resolution operator.

```
class student
{ int roll-no ;
  char name[10];
public : void reall()
         { - - };
       void show ()
         { - - };
};
```

03/09/2019

PAGE :
DATE : / /

CONSTRUCTOR

* Constructors are special methods (functions) that are automatically executed when an object of a class is created.
* It is special because its name is same as the class-name.
* They do not have any return type, not even void.
* Like other C++ functions they can have default arguments.
* There are 3 types of constructors :
  i) Default Constructor
  ii) Parameterized Constructor
  iii) Copy Constructor

* DEFAULT CONSTRUCTOR :
A constructor that accepts no parameter is called default constructor.
This Constructor is executed when an object of the class is created.
If we don't define any of the constructor then the default constructor is provided by the C++.

* PARAMETERIZED CONSTRUCTOR :
It is executed when an object of the class is created and is initialised with some values at the time of creation.

* COPY CONSTRUCTOR :
This constructor is executed when an object of the class is created and it is initialized with some other objects of the same class at the time of creation.

** DESTRUCTOR = ~ MONEY();
releases Memory of the Constructors defined.

EXAMPLE

PAGE :
DATE : / /

```
class MONEY
{   int rs;                          ] DATA MEMBERS
    int paisa;
public:
    MONEY ();                        // DEFAULT CONSTRUCTOR
    MONEY (int r, int p);            // PARAMETERIZED CONSTRUCT.
    MONEY (MONEY & M);               // COPY CONSTRUCTOR
~MONEY(); void read ();              ] MEMBER FUNCTIONS
          void show ();
};
MONEY :: MONEY ()                                  - POLYMORPHISM
{ rs = paisa = 0;
}
MONEY :: MONEY (int r, int p)
{ rs = r;
  paisa = p;
}
MONEY :: MONEY (MONEY & M)
{ rs = M.rs;                     // M's rs copies in MONEY's rs
  paisa = M.paisa;               // M's paisa copies in MONEY's paisa.
}
void MONEY :: read ()
{ Cin >> rs >> paisa;
}
void MONEY :: show ()
{ Cout << rs << paisa;
}
MONEY :: ~MONEY ()
{                                // Definition of Destructor is
}                                   always kept empty.
```

```
void main ()
{ MONEY M1 M4;
  cout << " First Amount is : ";
   M1.show();                    // OUTPUT = 0 0
  MONEY M2 (100,20);
  cout << " Second Amount is : ");
  M2.show();                     // OUTPUT = 100 20
  MONEY M3 (M2);
  cout << " third Amount is : ";
  M3.show();                     // OUTPUT = 100 20
  cout << " Enter an Amount : ";
  M4.read();
  M4.show();
  getch();
}
```

09
19

* CONSTRUCTOR OVERLOADING :

same as Polymorphism but with use of Constructors.

* INLINE FUNCTIONS :

Those functions whose definition ends in one line only.

Ex :

```
inline int sum ( int a, int b)
    { return a+b ; }
```

definition is replaced not transfered ← sum (3, 4);

- To eliminate the cost of functions, C++ proposed a new feature called Inline Function. In this, the compiler replaces the function calling with the corresponding function Code (Definition).

```
inline double cube (double a)
 { return (a * a * a) ; }
```

- All inline functions must be defined before they are called.

```
# include <iostream>
# include <conio.h>
//   inline float mul (float x, float y)
     { return x*y; }
     inline double div (double p, double q)
     { return p/q; }
     void main ()
     { float a = 12.34;
       float b = 9.82;
       cout << mul (a, b);
       cout << div (a, b);
       getch ();
     }
```

ARRAY OF OBJECTS

* The array of type class contains the objects of the class as its individual elements.
* We can also have array of variables that are of the type CLASS.

```
class Employee
{ char name [10];
  int age;
  public : void input ();
           void output ();
};
void Employee :: input ()
{ cin >> name >> age;
}
void Employee :: output ()
{ cout << name << age;
}
```

* An array of objects, all of whose elements are of the same class, can be declared just as an array of built in type.

```
void main()
{ Employee Manager[4];          → DECLARATION OF ARRAY OF OBJECT
  for (int i=0; i<4; i++)
    { cout << "Detail of Manager" << i+1;
      manager[i].input();
    }
  for (i=0; i<4; i++)
    { cout << "Details are:";
      Manager[i].output();
    }
  getch();
}
```

FRIEND FUNCTION

* this is declared using the keyword "friend".
* A Friend Function is a function that is not a member of a class but it is granted all the rights to access private part of the Class.
* A member function can also be declared as Friend of another class.

```
#include <iostream>
#include <conio.h>

class Person;                    // declaration of class Person
class Money
{ int rs;
  int paisa;
  public : void read()                    //
           { cin >> rs >> paisa; };
```

// cin >> P.name >> M.rs >> M.paisa ;

PAGE:
DATE: / /

```
        void show ()                                    //
        { cout << rs << paisa ; }
Declaration
FRIEND
FUNCTION →  friend void display ( Person P, Money M ) ;
                                  ↑          ↑
                                  arguments of class
                                  are passed
};
Class Person
{ char name [10];
  public : void input ();
           { cin >> name ; }
           void output ()
           { cout << name ; }
           friend void Display (Person P, Money M);
};
void Display ( Person P, Money M )
{ cout << P.name << "Contains" << M.rs << M.paisa;
}
void main ()
{ Money M1;
  M1.read ();
  Person P1;
  P1.input ();
  display (P1, M1);
  getch ();
}
```

09
10
19

UNIT - 3

PAGE:
DATE: / /

OPERATOR OVERLOADING

* C++ tries to make the user defined datatypes to behave in the same way as the build (pre) or built in datatypes
* OO provides a flexible option for the creation of the new definition for most of the C++ operators. Some of the operators that cannot be overloaded in C++ are as following:
  i) Scope Resolution operator (::)
  ii) Ternary operators CONDITIONALS (?:)
  iii) sizeof operator (sizeof)
  iv) Class member access operator (.*).

* **RULES of OPERATOR OVERLOADING:**
  i) We cannot use an operator in a manner that changes the syntax for the original operator.
  ii) We cannot create new operator symbols.

* **DEFINING OPERATOR OVERLOADING:**

returnType classname :: operator op (argument list)
{ - -
- - }

KEYWORD ↓
OPERATOR SYMBOL

** **Overloading UNARY OPERATORS:**

```
class Space
{ int x;
  int y;
  public : void input (int a, int b)
           { x = a;
             y = b;
           }
```

PAGE :
DATE : / /

```
void display () or void output ()
{ cout << x << y ;
}
void operator (-) () ;        → OPERATOR SYMBOL
                              → DECLARATION
};
void Space :: operator - ()
{   x = -x ;
    y = -y ;
}
void main ()
{ Space s1 ;
  s1. input (10, -20) ;
  s1. output () ;
  - s1 ;                      → CALLING OF OPERATOR FUNCTION
  s1. output () ;
  getch () ;
}
```

DEFINITION OF - OPERATOR OVERLOADING

* **overloading UNARY OPERATOR using FRIEND FUNCTION.**

```
class Space
{ int x ;
  int y ;
  public : void input (int a, int b)
           { x = a ;
             y = b ;
           }
           void output ()
           { cout << x << y ;
           }
```

DECLARATION OF FRIEND FUNCTION ← `friend void operator - (Space &s) ;`

```
};
```

COPIED OBJECT ↑

PAGE:
DATE: / /

```
void operator - (Space &s)
{ s1.x = -s1.x;
  s1.y = -s1.y;
}
```

NOTE : The Function Operator - () takes no arguments still it changes the sign of the data members of the object s1. Since this function is a member function of the same class, it can directly access the members of the object which activated it.

↳ FOR 1st PROGRAM.

** Overloading BINARY OPERATORS :

```
class complex
{ float x;
  float y;
  public: complex (float real, float imag)
          { x = real;
            y = imag;
          }
          complex operator + (complex &c);
          void display ()
          { cout << x << y; }
};
complex complex :: operator + (complex &c)
{ complex temp;
  temp.x = x + c.x;
  temp.y = y + c.y;
  return temp;
}
```

constructor

complex (class) used as a return type

display function

return type → complex

class name → complex

function name → operator +

temp object of complex class.

arguments passed.

PAGE :
DATE : / /

```
void main ()
{ complex C1, C2, C3 ;
  C1 = complex (2.5, 3.5) ;   or C1.complex (2.5, 3.5);
  C2 = complex (7.5, 12.0) ;
  C3 = C1 + C2 ;
  C1. display () ;
  C2. display () ;
  C3. display () ;
  getch () ;
}
```

C3 = C1.operator + (C2); working like this

+ function

argument for + function

values in + function (x and y)

* OVERLOADING BINARY OPERATOR using FRIEND FUNCTION :

```
class complex
{ float x ;
  float y ;
  public : complex (float real, float imag)
           { x = real;
             y = imag ;
           }
  friend complex operator + ( Complex &C1, complex &C2)
  void display ()
  { cout << x << y ; }
};
Complex operator + ( complex &C1, complex & C2)
{ complex temp ;
  temp.x = C1.x + C2.x;
  temp.y = C2.y + C1.y ;
  return temp ;
}
```

16.10.2019

TYPE CONVERSION

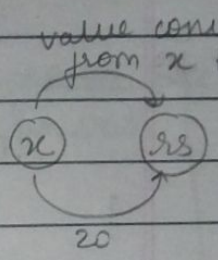
PAGE:
DATE: / /

* 3 Types of Conversions are there:
  i) Conversion from Basic dataType to Class type.
  ii) Conversion from Class type to Basic dataType.
  iii) Conversion from Class type to Class type.

* CONVERSION FROM BASIC to CLASS TYPE:

```
class Money
{ int rs;
  public : Money (int x)
           { rs = x; }
};
void main ()
{ Money M1;
  M1 (20);
}
```

rs = x; → Conversion



* CONVERSION FROM CLASS TYPE to BASIC TYPE:
  C++ allows us to define an overloaded casting operator that could be used to convert a class type data to a basic type.
  SYNTAX for creating casting operator:

```
operator type-name ()
{          };
```

→ in which you want to convert.

* PROGRAM:

```
class student
{ int roll_no;
  public : student (int x)
           { roll_no = x+10; }
  operator float ()
```

// CASTING OPERATOR DEFINITION.

PAGE:
DATE: / /

```
{ return float (roll-no)/2 ;
}
void show ()
{ cout << roll-no ; }
};
  void main ()
{ float f ;                    → float declaration.
   student s1(20);             → Parameterized Construction.
   s1. show ();
   f = s1;                     ⇒ OBJECT → BASIC.
   cout << f ; getch ();
}
```

// casting or type conversion.

value of float () returned to f.

* CONVERSION FROM CLASS TYPE to CLASS TYPE :
creating casting operator function.

```
class minute                   → SOURCE CLASS
{ int m;
  public : minute (int x)      → to be converted in which class.
           { m = x ; }
  operator hour ()
           { hour h1 ;
             h1.h = m/60 ;
             return (h1) ;
           }
```

CASTING FUNCTION

```
  void show ()
  { cout << " Minutes = " << m ; }
  };
class hour                     → DESTINATION CLASS.
  { int h ;
```

PAGE:
DATE: / /

```
public : hour()
         { h = 0; }
         }
         void show()
         { cout << h; }
};
void main()
{ minute min[60];
  hour hr;
  hr = min;
60//  min.show();
1//   hr.show(); getch();
}
```

default constructor

→ TRANSFERRING OR CONVERTING VALUES.

UNIT 3 finished

17.10.2019

UNIT-4

INHERITANCE

PAGE :
DATE : / /

* **Single level Inheritance :**

```
class A
{        };
class B : public A        -> INHERITING CLASS A to class B
{        };
void main()
{   getch();  }
```

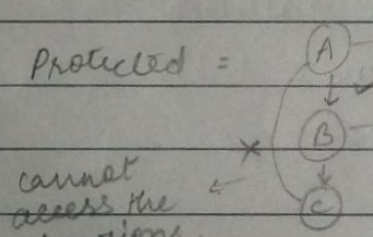
(VISIBILITY MODE)

- Three visibility Modes:

i) Public ii) Private iii) Protected.

Visibility Mode / Class Member	PRIVATE	PUBLIC	PROTECTED
* PRIVATE	Never Inherited	never Inherited	never Inherited
* PROTECTED	Private	protected	Protected
* PUBLIC	Private	Public	Protected

- Protected =



- Class Member = Both functions.

Ex :

```
class one                    -BASE CLASS.
{ int x;
  public : void readx()
           { cin >> x; }
```

PAGE:
DATE: / /

```
void showx
 { cout << x ;
 }
};
class two : public one                    - DERIVED CLASS
 { int y ;
   public : void readxy ()
            { readx () ;  cin >> y ;
            }
            void showxy ()
            { showx () ;
              cout << y ; }
};
 void main ()
{ two t ;
   t. readxy () ;
   t. showxy () ;  getch () ;
}
```

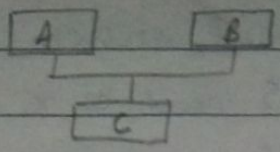
→

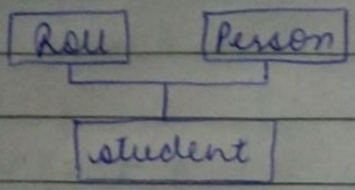
```
class one
{ protected : int x ;
  public : same as above ;
};
class two : public one
{ protected : int y ;
  public : void readxy ()
           { cin >> x ;  cin >> y ; }
           void showxy ()
           { cout << x ; cout << y ; }
};
```

x (one) ↓ protected x (two)

PAGE :
DATE : / /

* Multiple Inheritance :






```
class roll
{ protected : int r ;
  public : void read ()
           { cin >> r ; }
           void show ()
           { cin >> r ; } cout << name ;
};
class Person
{ protected : char name [20] ;
  public : void readn ()
           { cin >> name ; }
           void shown ()
           { cout << name ; }
};
```

NO need

NO NEED

into this ← from these → INHERITANCE

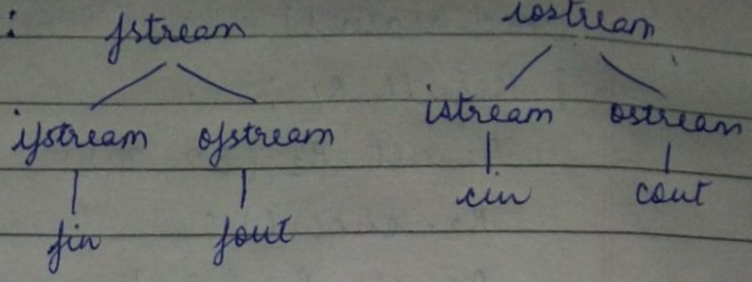
```
class student : public roll, public person
{ protected : int marks ;
  public : void reads ()
           { cin >> r >> name >> marks ; }
           void shows ()
           { cout << r << name << marks ; }
};
void main ()
{ student s1 ;
  cout << "Enter details : " ;
  s1.reads () ; s1.shows () ; getch () ; }
```

14.11
2019

UNIT - 5

PAGE :
DATE : / /

* **FILE HANDLING :**



- for file handling : <fstream.h>

	Type	Description
i)	fstream	- This type represents the file stream which has the capabilities of both ofstream & ifstream which means it can create file, write info. to the file and read info. from the file.
ii)	ifstream	- This type represents input file stream & it is used to read info. from the file.
iii)	ofstream	- This type represents output file stream & it is used to create file and write info. to the file.

- fstream → read, write, create
- ifstream → read info
- ofstream → create file / write info.

Q: WAP to store 10 Numbers in a file ?

→
```
#include <iostream>          #include <fstream.h>
using namespace std;
int main()
{ fstream file;
  file.open("ABC.txt");
  int i, value;
  for (i=0; i<10; i++)
  { cin >> value; file << value << endl; }
  file.close();
}
```

PAGE:
DATE: / /

* **Opening a file :**
A file must be opened before you can read from it or write to it. Either ofstream / fstream may be used to open a file.
Syntax :

```
void open ("FILENAME", MODENAME);
```

* **Closing a file :**
When a C++ program terminate it automatically closes all the streams and release all the allocated memory and close all the opened files.
Syntax :

```
void close ();
```

Q: WAP to read numbers from the above created file and display the output on the console screen.

→

```
#include <iostream.h>   #include<fstream.h>
using namespace std;

int main ()        / void main ()
{ fstream file;
  file.open ("ABC.txt");
  int i, value;
  for (i=0; i<10; i++)
  { file >> value;
    cout << value << endl;
  }
  file.close ();   getch ();
}
```