- intro C++ features of C++
- history
- need
- tokens
  - identifier, datatypes, keywords, operators, const, literals, string, esc seq

- control structure
  - condition structure
  - loop (for, do, do..while)
  - switch..case

- user defined functions - declaration, definition and call

- pointers (types - null, wild, dangling, void, pointer to pointer, array of pointers, pointer to array, function pointer)

- array (pointer arithmetics, how to access elements)
  1-d, 2-d, 3-d - on stack and on heap

- class (OOP)

- data members, member function

- owner of class, user of class

- private, public

- setters(modifiers) and getters

- this pointer

---

function overloading -

We can write one function many times, but function signature for each function should be different.

We can use same function name to define more than one functions with different signatures

signature of a function means type and number of parameters

name decoration / mangling is done by compiler at the time compilation

name mangling means all the overloaded functions are given different names at the time of compilation. compile time binding of functions is done.

```
int _Z0sumii(int, int);      //#1
float sum(float, float);  //#2
float sum(int, float);    //#3
int sum(int, int, int);   //#4
float sum(int, float, float);  //#5
float sum(float, int)


float sum(int, int);  //  not allowed  funtion #1 matches with this



#include<iostream>
using namespace std;

int _Z0sumii(int, int);      //#1
float _Z1sumff(float, float);  //#2
float sum(int, float);    //#3
int sum(int, int, int);   //#4
float sum(int, float, float);  //#5
float sum(float, int);

int main()
{
   cout<<"\n  int sum(int, int) = "<<_Z0sumii(10,12);
   cout<<"\n float sum(float, float) = "<<_Z1sumff(10.0f,12.0f);
   cout<<"\n float sum(int, float) = "<<sum(10,12.75f);
   cout<<"\n int sum(int, int, int) = "<<sum(10,12,15);
   cout<<"\n float sum(int, float, float) = "<<sum(10,12.0f,15.0f);
   cout<<"\n float sum(float, int) = "<<sum(10.0f, 12);
   return 0;
}

int _Z0sumii(int a, int b)
{
   return a + b;
}

float _Z1sumff(float a, float b)
{
   return a + b;
}

float sum(int a, float b)
{
    return (float)a + b;
}
int sum(int a, int b, int c)
{
    return a + b + c;
}

float sum(int a, float b, float c)
{
    return a + b + c;
}

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

---

operator overloading - Giving meaning to operators in user defined class, so that the objects of that class should support exp with operators.

```
 +, -, *, /, %, >, <, >=, <=, ==, =,

 ++, --, +=, -=, *=, /=, %=, >>, <<, [ ],


 &, *, new, delete, sizeof, . , ->, ::, ? :


 &&, ||, !,
```

```
#include<iostream>
using namespace std;

class mycomplex{
 int real, imag;

 public:
  // mycomplex() { }
  mycomplex()       //default constructor
  {
     cout<<"\n Default constructor is called....";
     real = 0 ;
     imag = 0;
  }

  mycomplex(int r)
  {
     cout<<"\n single parameter constructor is called....";
     real = r;
```

```
      imag = r;
   }

   mycomplex(int r, int i)
   {
      cout<<"\n two parameter constructor is called...";
      real = r;
      imag = i;
   }

   void setReal(int r)
   {
       real = r;
   }
   void setImag(int r)
   {
      imag = r;
   }
   void display()
   {
      cout<<real<<" + "<<imag<<"i\n";
   }

   mycomplex add(mycomplex c)
   {
     complex res;
     res.real = real + c.real;
     res.imag = imag + c.imag;

     return res;
   }

   ~mycomplex()
   {
     cout<<"\n Destructor is called....";
   }



};


int main()
{
    mycomplex c1(4), c2(2,6), c3;
    //c1.setReal(4);
    //c1.setImag(5);
    //c2.setReal(10);
    //c2.setImag(6);

    c1.display();
    c2.display();
    c3.display();
    c3 = c1.add(c2);

    return 0;
}
```

---

constructor and destructor

constructor-
1. member function of class
2. constructor function name is same as class name
3. called implicitly whenever the object is created (need not to be called)

4. We can overload constructors
5. We write instrcutions for memory allocation for member varaibles and initialization of member variable.
6. If constructor is not given in the class, compiler provides default constructor for each class
7. If any type of constructor is given in class, compiler will not provide default constructor.
8. does not have any return data type, not even void
9. It has to be declared as public.

destructor -

1. member function of class
2. public
3. name of function is ~nameofClass()
4. takes no arguments
5. no return data type
6. one class only one destructor (overloading is not possible)
7. will be called whenever the object goes out of scope.
8. will release resources aquired by object

---

write constructor, parameterized constructor and destructor for date class

```
write operator overloading  + with int value
```

```
class point {

  int x, y;

  public:
     constructor  ---    ??

     destructor

     setter and getters for all members

     display()    (x , y)
};

int main()
{
    create objec of point and use functionalities
}
```

```
class student
{   int rno;
    //char name[20];
    //char branch[30];
    int marks;
    char grd;

   public :
     constructors   -   default, parameterized
     destructor

     getter and setter for all members

     display
};
```

```
class Integer{
   int *ptr;

  public:

    Integer() { ptr = new int;  *ptr = 0; }

    Integer(int n) {
        cout<<"\n parameter constructor is called...";
        ptr = new int;
        *ptr = n;
    }

    ~Integer() { cout<<"\n Destructor is called..."; delete ptr; ptr = NULL; }

    Integer(Integer const &I)
    {
        cout<<"\n Copy constructor is called...";
        ptr = new int;
       *ptr = *I.ptr;
    }

    void display()
    {
        cout<<"\n Value = "<<*ptr;
    }
};


int main()
{
   Integer obj1(5);
   Integer obj2(obj1);

   obj1.display();
   obj2.display();
   return 0;
}
```