CONSTRUCTORS AND DESTRUCTORS

- Member functions cannot be used to assign the initial values to the member variables at the time of creation of their objects.
- Initialize the objects when created and destroy when they are no longer necessary.
- C++ provides special member functions called the constructor which enables an object to initialize itself when it is created. This is known as automatic initialization of objects.
- Another member function destructor destroys the objects when they are no longer required.

* Reference: Object oriented Programming with C++ -E Balaguruswamy ,TMH publications 6th edition,2015

CONSTRUCTORS

- A special member function whose function is to to initialize the objects of its class.
- It's name is same as the class name.
- Constructor is invoked whenever an object of its associated class is created
- It is called as constructor because it constructs the values of data members of the class.
- Ex. :

```
class integer
{ int m,n;
Public: integer (void);  //constructor declared
.....
};
Integer::integer(void)      //constructor defined and initialized all parameters to
                            //zero
{
m=0;n=0;
}
```

- Integer int1; int1 object of type integer is created and initializes its data members m and n to zero
- This will be helpful when a large number of objects are there.
- A constructor that accepts no parameters is called a default constructor.

- Special characteristics of constructors:
  - They should be declared in public section
  - They are invoked automatically when the object is created
  - They do not have return type and hence cannot return values
  - They cannot be inherited though a base class can call the base class constructor
  - They can have default arguments like other functions
  - Constructors cannot be virtual
  - Their addresses cannot be referred to.
  - They make implicit calls to the operators 'new' and 'delete' when the memory allocation is required.
  - When a constructor is declared for a class, initialization of the class objects become mandatory.

Parameterised constructors

- Constructors that can take arguments are called parameterized constructors.
- Initializes various data elements of different objects with different values when they are created.
- We can pass arguments to the constructor function when the objects are created
- Ex. :

```
class integer
{ int m, n;
Public: integer (int, int);   //constructor declared
.....
};
Integer::integer(int x, int y)     //constructor defined and initialized parameters
{
m=x; n=y;
}
```

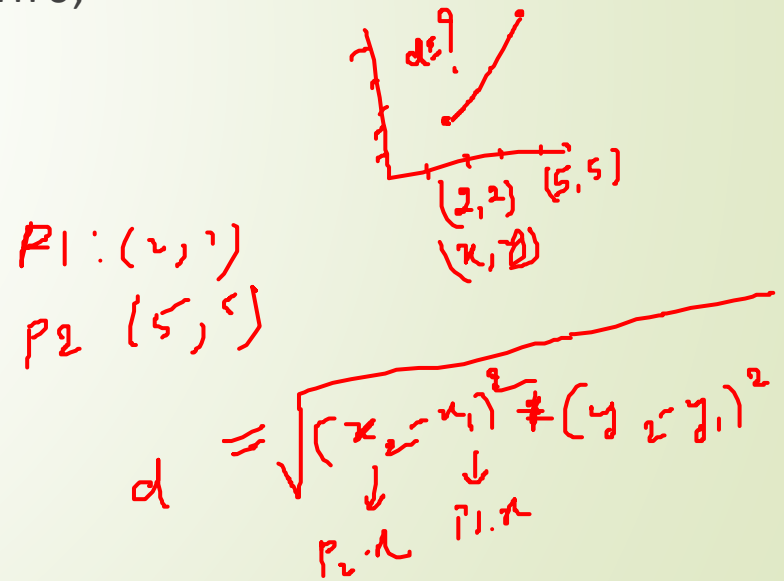
- The object declaration statement will be:
  - By calling the constructor explicitly- Ex. integer int1=integer (0,100);
  - By calling the constructor implicitly (short hand method)- Ex. integer int1(0,100);

```
class integer
{ int m,n;
Public:
integer(int x, int y)     // inline constructor
{
m=x; n=y;
}
};
class integer
{Public:
integer(integer);         // illegal-parameters of a class cannot be the same type
                          //as the class
};                        // illegal
```

```
class integer
{Public:
integer(integer&);        // legal, can accept a reference to its own class as a
                          //parameter
};                        // Such constructor is called copy constructor
```

```
// Example to illustrate the
concept of constructors

# include <iostream.h>

class point

{

int x,y;

public:

point (int a,int b)

{

 x=a;

x=b;

}

void display()

{

cout<<"Point :
("<<x<<","<<y<<")"<<endl;

}

};
```

```
int main()

{

point p1(2,2);

point p2(5,5);

p1.display();

p2.display();

getch();

return 0;

}
```



Multiple constructors in a class

Ex.:

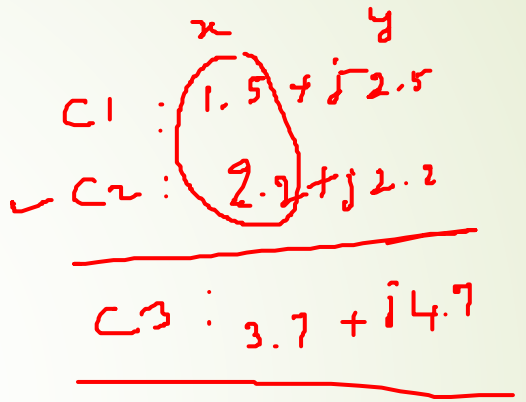
```
class integer
{
 int m,n;
Public:
integer(){m=0;n=0;}
integer (int a, int b)
{
m=a; n=b;
}
Integer(integer & i)        //receives one integer object as an argument
{
m=i.m;n=i.n;}
};
```

The declarations would be:

- integer I1; //would invoke the first constructor
- integer I2(20,40); //would invoke the second constructor
- integer I3(I2); // would invoke the third constructor and copies the values of I2 to I3.
- It sets the value of every data element I3 to the value of the corresponding data element of I2. Such a constructor is called the ***copy constructor***.
- When more than one constructor function is defined in a class, **constructor is said to be overloaded.**

Ex.:

```
#include<iostream.h>
class complex
{
 float x,y;
public:
complex(){}
complex(float a) {x=y=a;}
complex(float real, float imag)
{ x=real; y=imag;}
friend complex sum(complex,complex);
friend void show(complex);
};
complex sum(complex c1,complex c2)
{
complex c3;
c3.x=c1.x+c2.x;
c3.y=c1.y+c2.y;
return(c3);
}
```



```
void show(complex c)
{ cout<<c.x<<"+j"<<c.y<<endl;
}
int main()
{
complex A (1.5,2.5);   //same as complex A; A=complex (1.5,2.5)
complex B(2.2);
complex C;
C=sum(A,B);
cout<<"A=";show(A);
cout<<"B=";show(B);
cout<<"C=";show(C);
return 0;
}
```

- Constructors with default arguments
  - Ex.: complex(float real,float imag=0);
  - Statement can be complex C(5.0); //real=5.0 imag=0

Dynamic initialization of objects

- The initial value of the object can be provided during run time.
- Flexibility of using different format of data at run time depending on the situation.

```
#include <iostream.h>
#include <math.h>
#include <conio.h>
const float pi=3.14;
class area
{
private:
int a,b;
float r;
float AREA;
public:
area(){}
area(int x,int y)
{a=x;b=y;
AREA=x*y;
}
area(float z)
{r=z;
AREA=pi*r*r;
}
void display(void)
{
cout<<"Area is :"<<AREA<<"sq
mt"<<endl;
}
};
```

```
int main()
{
area REC,CIR;
int l,b;
 float h;
 cout<<"enter the length and breadth of rectangle"<<endl;
 cin>>l>>b;
REC=area(l,b);
REC.display();
 cout<<"enter the radius of circle"<<endl;
 cin>>h;
 CIR=area(h);
 CIR.display();
 getch();
 return(0);
}
```

- **Copy constructor**
  - A copy of the constructor is used to declare and initialize an object from another object.
  - Ex.: integer (integer &i);
  - Integre I2(I1); //define object I2 and initialize it to the values of I1
  - Integer I2=I1; //correct
  - I2-I1 ; is legal if I1 and i2 are objects; value of I1 is assigned to I2 member by member; here = operator is overloaded.

- **Dynamic constructors**
  - Allocation of memory to objects at the time of their construction is knoen as dynamic construction of objects
  - Memory is allocated with the help of the 'new' operator

- **Destructor**

  - Used to destroy the objects that have been created by a constructor
  - It is a member function, whose name is the same as the class name but is preceded by a ~ (tilde)

  Ex.: ~integer(){}

  - A destructor does not take any value nor does it returns any value
  - It will be invoked implicitly by the compiler upon exit from the program.
  - It is a good practice to declare destructors in a program since it releases memory space for future use.

```
#include<iostream>
using namespace std;
int count=0;
class A
{
public:
A()
 {
 count++;
 cout<<"No. of objects created:"<<count<<endl;
 }
 ~A()
 {
 cout<<"No. of objects
destroyed:"<<count<<endl;
 count--;
 }
};
int main()
{
 cout<<"Enter main";
 A a1,a2,a3,a4,a5,a6,a7,a8;
 {
 cout<<"Enter block1"<<endl;
 A a9;
 }
 cout<<"Re-enter
main"<<endl;
 return 0;
}
```

1+1

1
2
3
.
.
.
Block 1

- Define a class string that works as user defined string type. The class should include an uninitialized constructor and also include a constructor to initialize an object with string constant at the time of creation. Include a function that adds two strings.

Write a complete program to test the class to perform the following tasks:

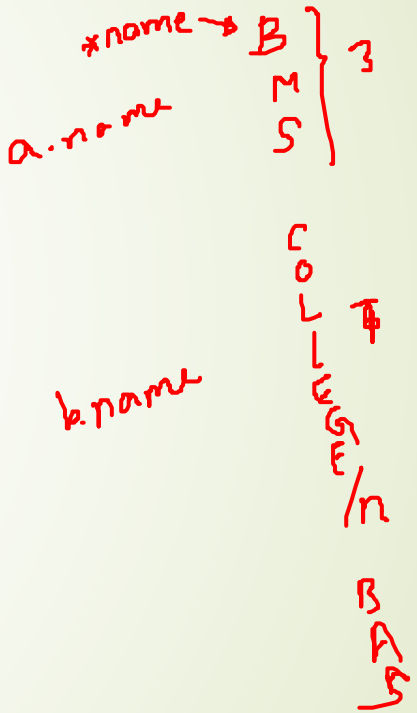
a) Creates uninitialized class

b) Creates objects with string constants

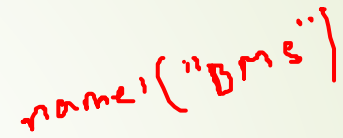
c) Concatenates two strings properly

d) Displays a desired string objects

```
#include<iostream.h>
using namespace std;
class string
{
char *name;
int len;
public:
string()
{len=0;
name=new char[len+1];
}
string (char *s)
{
len=strlen(s);
name=new char[len+1];
strcpy(name,s);
}
```



```
void display(void)
{
cout<<name<<endl;}
void join(string &a, string &b);
};
void string::join (string &a,string &b)
{
len=a.len+b.len;
delete name;
name=new char[len+1];
strcpy(name,a.name);
strcat(name,b.name);
};
```

len = 10
name = ''

```
int main()
{
char *first="BMS";
string name1(first), name2("college"),
name3("Basavanagudi"),s1,s2;
s1.join(name1,name2);
s2.join(s1,name3);
name1.display();
name2.display();
name3.display();
s1.display();
s2.display();

 return 0;
}
```