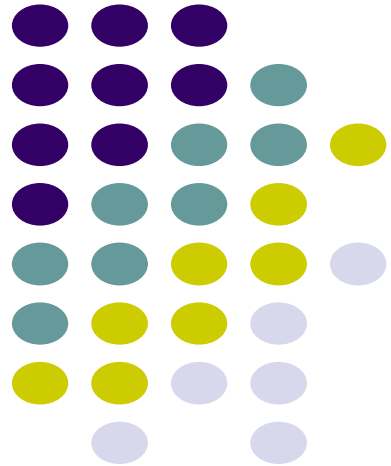
Object Oriented Programming in C++

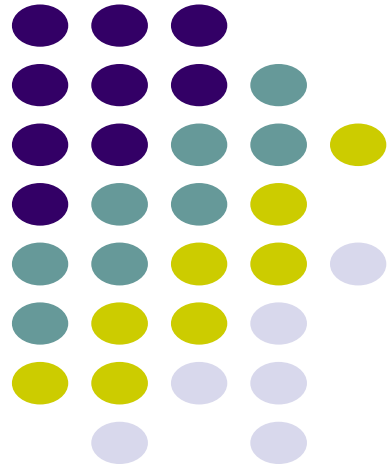
Presented by Bhimashankar T

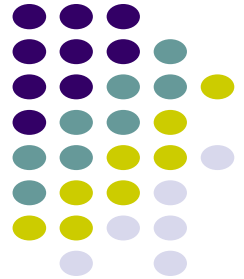
E-Mail: bhima.t@gmail.com

PhNo:+91 9980156833 / 6363863430



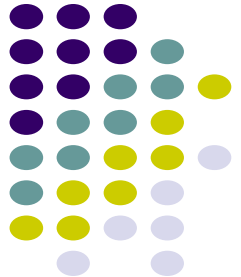
DAY 01





OOPS Concept

- Objects
- Classes
- Abstraction
- Encapsulation
- Inheritance
- Overloading
- Exception Handling

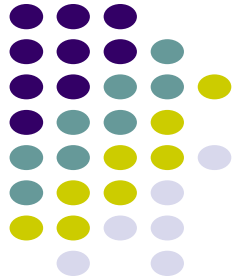


Introduction

- C++ is a statically typed, compiled, general-purpose, case-sensitive, free-form programming language that supports procedural, object-oriented, and generic programming.
- C++ is regarded as a middle-level language, as it comprises a combination of both high-level and low-level language features.
- C++ was developed by Bjarne Stroustrup starting in 1979 at Bell Labs in Murray Hill, New Jersey, as an enhancement to the C language and originally named C with Classes but later it was renamed C++ in 1983.
- C++ is a superset of C, and that virtually any legal C program is a legal C++ program.

Note: A programming language is said to use static typing when type checking is performed during compile-time as opposed to run-time.

Difference between Static typing and Dynamic typing



Static Typing

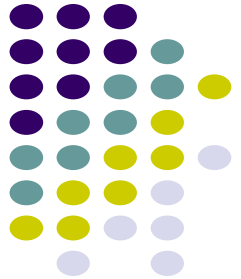
Static typing is a typing system where variables are bound to a data type during compilation. Once a variable is assigned a data type it remains unchanged throughout the programs execution. This binding promotes type safety and detects errors at an early stage.

Examples of Statically Typed Languages are C++, Java, Rust

Dynamic Typing

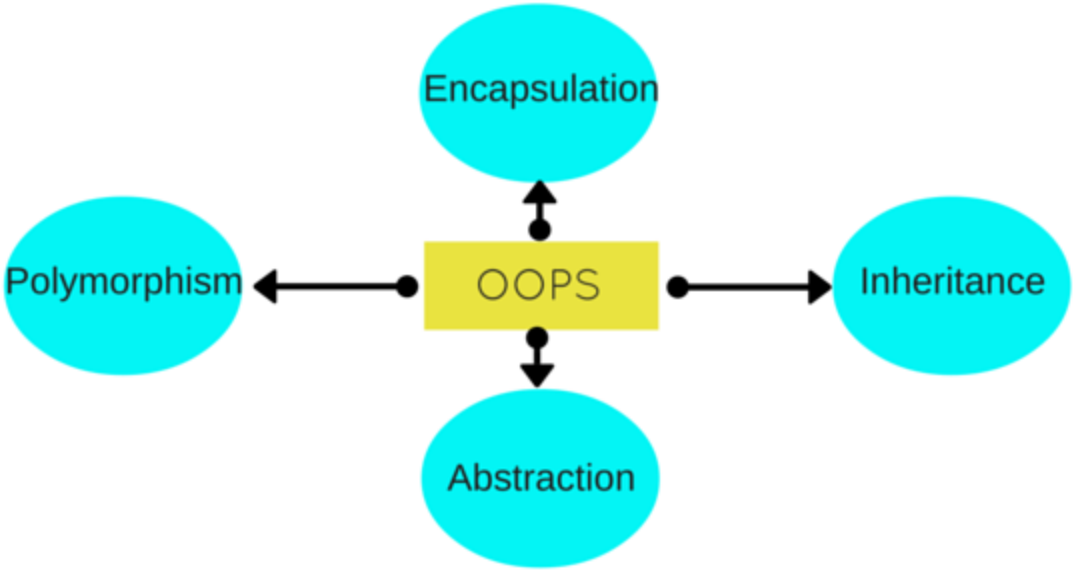
Dynamic typing allows variables to be bound to data types at runtime instead of during compilation. This flexibility enables concise code and ease of use. It compromises on type safety as a trade-off..

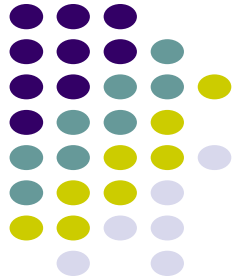
Examples of Statically Typed Languages are Python, JS, Ruby



Object Oriented Programming

Object Oriented programming is a programming style that is associated with the concept of Class, Objects and various other concepts revolving around these two, like Inheritance, Polymorphism, Abstraction, Encapsulation etc.





OOPS Concept Definitions

Objects

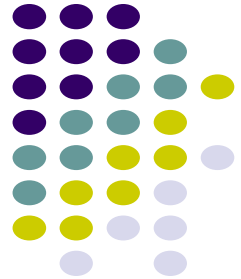
Objects are the basic unit of OOP. They are instances of class, which have data members and uses various member functions to perform tasks.

Class

It is similar to structures in C language. Class can also be defined as user defined data type but it also contains functions in it. So, class is basically a blueprint for object. It declare & defines what data variables the object will have and what operations can be performed on the class's object.

Abstraction

Abstraction refers to showing only the essential features of the application and hiding the details. In C++, classes can provide methods to the outside world to access & use the data variables, keeping the variables hidden from direct access, or classes can even declare everything accessible to everyone, or maybe just to the classes inheriting it. This can be done using access specifiers.



OOPS Concept Definitions

Encapsulation

It can also be said data binding. Encapsulation is all about binding the data variables and functions together in class.

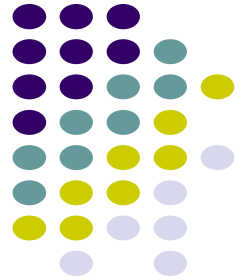
Inheritance

Inheritance is a way to reuse once written code again and again. The class which is inherited is called the **Base** class & the class which inherits is called the **Derived** class. They are also called parent and child class.

So when, a derived class inherits a base class, the derived class can use all the functions which are defined in base class, hence making code reusable.

Polymorphism

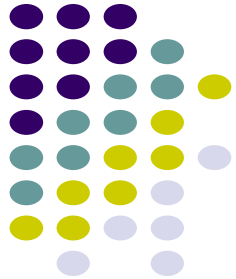
It is a feature, which lets us create functions with same name but different arguments, which will perform different actions. That means, functions with same name, but functioning in different ways. Or, it also allows us to redefine a function to provide it with a completely new definition.



OOPS Concept Definitions

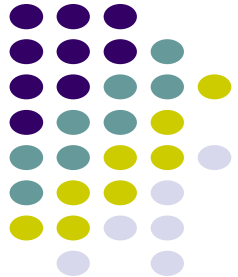
Exception Handling

Exception handling is a feature of OOP, to handle unresolved exceptions or errors produced at runtime.



Comparison between C and C++

C Programming	C++ Programming
Top-Down approach	Bottom-up approach
Function - driven	Object - driven
Cannot use functions inside structures	can use functions inside structures
Data is not secured	Data is hidden (secured)
Procedural oriented language	Procedural and Object oriented language
Cannot do overloading with C	Supports both operator and function overloading
main() need not return any value	main() should return a value

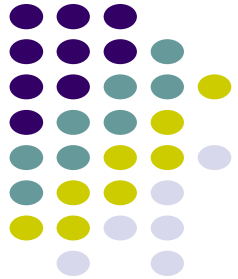


C++ Basic Syntax

C++ program can be defined as a collection of objects that communicate via invoking each other's methods.

Let us now briefly look into what do class, object, methods and instant variables mean.

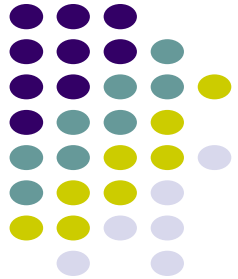
- **Object** - Objects have states and behaviors. Example: A dog has states - color, name, breed as well as behaviors - wagging, barking, eating. An object is an instance of a class.
- **Class** - A class can be defined as a template/blueprint that describes the behaviors/states that object of its type support.
- **Methods** - A method is basically a behavior. A class can contain many methods. It is in methods where the logics are written, data is manipulated and all the actions are executed.
- **Instant Variables** - Each object has its unique set of instant variables. An object's state is created by the values assigned to these instant variables.



C++ Program Structure

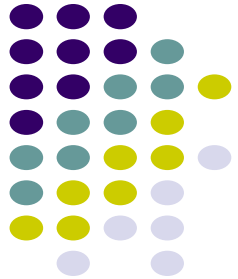
```
#include<iostream>
using namespace std;
// main() is where program execution begins.

int main()
{

   cout <<"Hello World"; // prints Hello World
   return 0;

}
```

C++ Program Structure



Let us look various parts of the above program:

- The C++ language defines several headers, which contain information that is either necessary or useful to your program. For this program, the header <iostream> is needed.
- The line using namespace std; tells the compiler to use the std namespace. Namespaces are a relatively recent addition to C++.
- The next line // main() is where program execution begins. is a single-line comment available in C++. Single-line comments begin with // and stop at the end of the line.
- The line int main() is the main function where program execution begins.
- The next line cout << "Hello World"; causes the message "Hello World" to be displayed on the screen.
- The next line return 0; terminates main( )function and causes it to return the value 0 to the calling process.

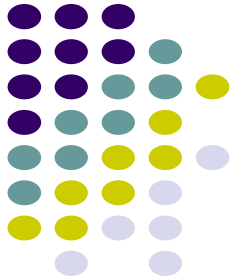


C++ Identifiers:

A C++ identifier is a name used to identify a variable, function, class, module, or any other user-defined item. An identifier starts with a letter A to Z or a to z or an underscore (_) followed by zero or more letters, underscores, and digits (0 to 9).

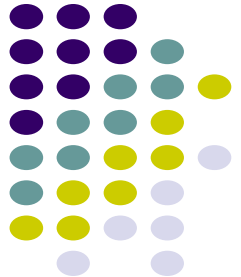
C++ does not allow punctuation characters such as @, $, and % within identifiers. C++ is a case-sensitive programming language. Thus, Manpower and manpower are two different identifiers in C++.

C++ Keywords:



The following list shows the reserved words in C++. These reserved words may not be used as constant or variable or any other identifier names.

Asm	else	new	this
Auto	enum	operator	throw
Bool	explicit	private	true
Break	export	protected	try
Case	extern	public	typedef
Catch	false	register	typeid
Char	float	reinterpret_cast	typename
Class	for	return	union
Const	friend	short	unsigned
const_cast	goto	signed	using
continue	If	sizeof	virtual
Default	inline	Static	void
Delete	int	static_cast	volatile
Do	long	Struct	wchar_t
Double	mutable	Switch	while
dynamic_cast	namespace	Template	

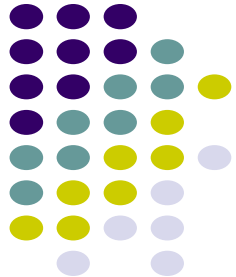


Data Types in C++

Already we have seen in C language. But some additional types are,

bool	Boolean (True or False)
void	Without any Value
wchar_t	Wide Character

Wide char and library functions in C++



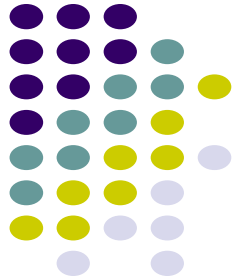
Wide char is similar to char data type, except that wide char take up twice the space and can take on much larger values as a result. char can take 256 values which corresponds to entries in the ASCII table. On the other hand, wide char can take on 65536 values which corresponds to UNICODE values which is a recent international standard which allows for the encoding of characters for virtually all languages and commonly used symbols.

Just like the type for character constants is char, the type for wide character is wchar_t. This data type occupies 2 or 4 bytes depending on the compiler being used.
Mostly the wchar_t datatype is used when international languages like Japanese are used.

```
// An example C++ program to demonstrate use of wchar_t
int main()
{
        wchar_t w = L'A';
        cout << "Wide character value:: " << w << endl ;
        cout << "Size of the wide char is:: " << sizeof(w);
        return 0;
}
```

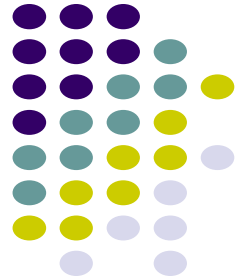
```
// An example C++ program to demonstrate use
// of wchar_t in array
#include <iostream>
using namespace std;

int main()
{
        // char type array string
        char caname[] = "geeksforgeeks" ;
        cout << caname << endl ;

        // wide-char type array string
        wchar_t waname[] = L"geeksforgeeks" ;
        wcout << waname << endl;

        return 0;
}
```

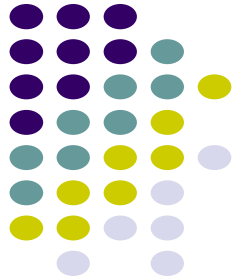
```
// An example C++ program to demonstrate use
// of wcslen()
#include <iostream>
#include<cwchar>
using namespace std;

int main()
{
        // wide-char type array string
        wchar_t waname[] = L"geeksforgeeks" ;

        wcout << L"The length of '" << waname
                << L"' is " << wcslen(waname) <<
endl;

        return 0;
}
```



```
// An example C++ program to demonstrate use
// of wcscpy()
#include <iostream>
#include<cwchar>
using namespace std;

int main()
{
        wchar_t waname[] = L"geeksforgeeks" ;
        wchar_t wacopy[14];
        wcscpy(wacopy, waname);
        wcout << L"Original = " << waname
                << L"\nCopy = " << wacopy << endl;

        return 0;
}
```

```
// An example C++ program to demonstrate use
// of wcscat()
#include <iostream>
#include<cwchar>
using namespace std;

int main()
{
        wchar_t string1[] = L"geeksforgeeks" ;
        wchar_t string2[] = L" is for Geeks" ;

        wcscat(string1, string2);

        wcout << L"Concatenated wide string is = "
                << string1 << endl;

        return 0;
}
```





Data Types in C++

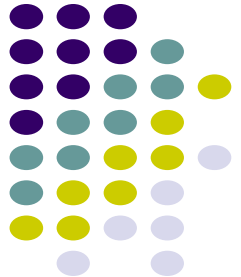
Enum as Data type

Enumerated type declares a new type-name and a sequence of value containing identifiers which has values starting from 0 and incrementing by 1 every time.

For Example :

enum day(mon, tues, wed, thurs, fri) d;

Here an enumeration of days is defined with variable d. *mon* will hold value 0, *tue* will have 1 and so on. We can also explicitly assign values, like, enum day(mon, tue=7, wed);. Here, *mon* will be 0, *tue* is assigned 7, so *wed* will have value 8.

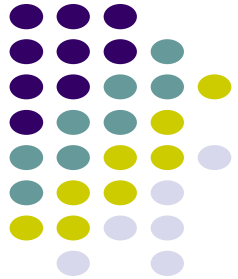


Data Types in C++

Modifiers

Specifiers modify the meanings of the predefined built-in data types and expand them to a much larger set. There are four data type modifiers in C++, they are :

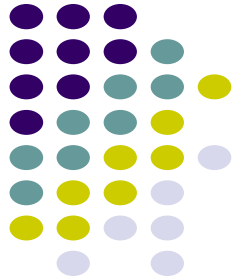
1. long
2. short
3. signed
4. unsigned



Type Qualifiers in C++

The type qualifiers provide additional information about the variables they precede.

Qualifier	Meaning
const	Objects of type **const** cannot be changed by your program during execution
volatile	The modifier **volatile** tells the compiler that a variable's value may be changed in ways not explicitly specified by the program.
restrict	A pointer qualified by **restrict** is initially the only means by which the object it points to can be accessed. Only C99 adds a new type qualifier called restrict.



Some special types of variable

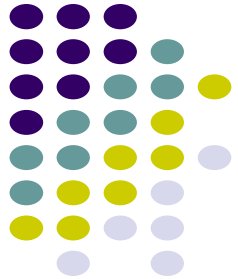
There are also some special keywords, to impart unique characteristics to the variables in the program. Following two are mostly used, we will discuss them in details later.

Final - Once initialized, its value cant be changed.

Static - These variables holds their value between function calls.

Example :

```
#include <iostream.h>
using namespace std;
int main()
{
        final int i=10;
        static int y=20;
}
```



Operators

These we have already learnt in C language

typedef and Pointers

typedef can be used to give an alias name to pointers also. Here we have a case in which use of typedef is beneficial during pointer declaration.

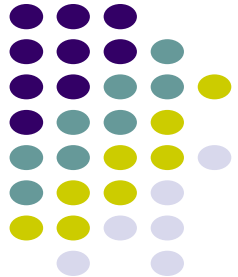
In Pointers * binds to the right and not the left.

```
int* x, y ;
```

By this declaration statement, we are actually declaring **x** as a pointer of type int, whereas **y** will be declared as a plain integer.

```
typedef int* IntPtr ;
IntPtr x, y, z;
```

But if we use **typedef** like in above example, we can declare any number of pointers in a single statement.

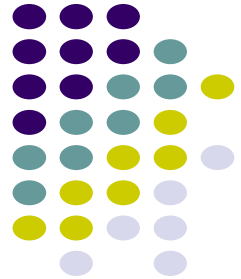


Branching and Looping Statements

1. Simple *if* statement
2. *If....else* statement
3. Nested *if....else* statement
4. *else if* statement
5. goto statement

Looping Statements

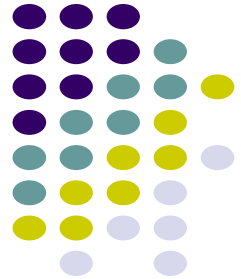
1. *while* loop
2. *for* loop
3. *do-while* loop



Storage Classes in C++

These are basically divided into 5 different types :

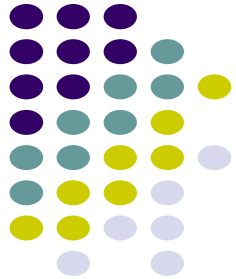
1. Global variables
2. Local variables
3. Register variables
4. Static variables
5. Extern variables
6. Mutable



Storage Classes in C++

Static Variables

```
void fun()
{
 static int i = 10;
 i++;
 cout << i;
}
int main()
{
 fun();      // Output = 11
 fun();      // Output = 12
 fun();      // Output = 13
}
```



Storage Classes in C++

Extern Variables

This keyword is used to access variable in a file which is declared & defined in some other file, that is the existence of a global variable in one file is declared using extern keyword in another file.

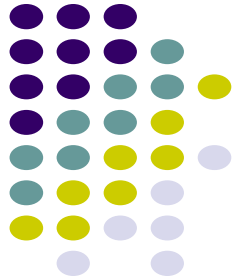
File1.cpp

```
#include<iostream>
using namespace std;
int globe ;
void func();
int main()
{
 . . . . . .
 . . . . .
}
```

File2.cpp

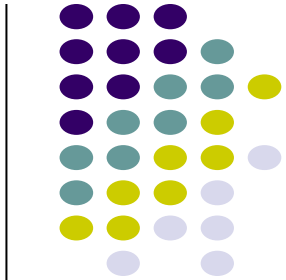
```
extern int globe ;
int b = globe + 10 ;
```

Global variable in one file is used in other file by **extern keyword**

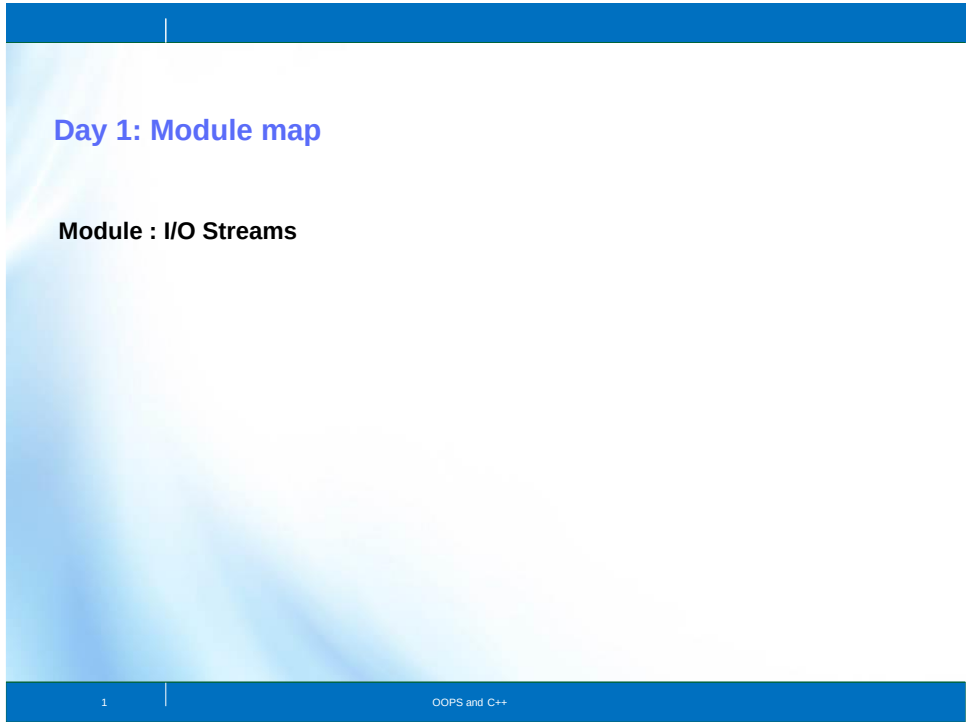


Mutable storage class in C++

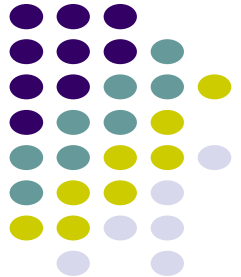
The mutable specifier applies only to class objects, which are discussed later in this tutorial. It allows a member of an object to override constness. That is, a mutable member can be modified by a const member function



IO Streaming



Bhimashankar Takalki

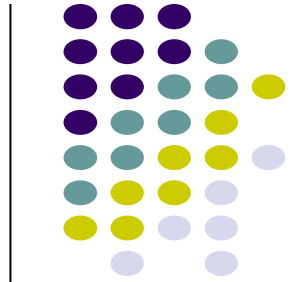


Functions in C++

Calling a Function

Functions are called by their names. If the function is without argument, it can be called directly using its name. But for functions with arguments, we have two ways to call them,

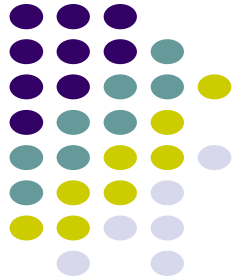
1. Call by Value
2. Call by Pointer
3. Call by Reference



Function - Call by value

```
#include <iostream>
using namespace std;
// function declaration
int max(int num1, int num2);
int main ()
{
 local variable declaration:
 int a = 100;
 int b = 200; int ret;
 //calling a function to get max value.
 ret = max(a, b);
 cout << "Max value is : " << ret << endl;
 return 0;
}
```

```
// function returning the max between two
numbers
int max(int num1, int num2)
{
      // local variable declaration
      int result;
      if (num1 > num2)
            result = num1;
      else
            result = num2;
 return result;
}
```

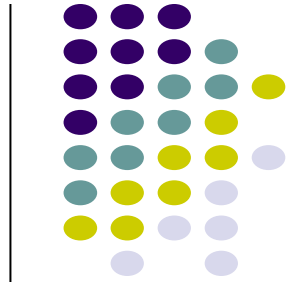


Functions in C++

Call by Pointer

```
void calc(int *p);
int main()
{
 int x = 10;
 calc(&x);     // passing address of x as argument
 printf("%d", x);
}

void calc(int *p)
{
 *p = *p + 10;
}
```

Output : 20



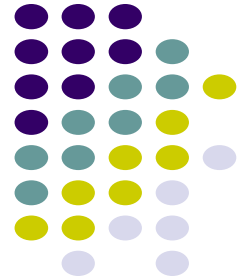
Functions in C++

Call by Reference

Call by reference :

```
void swap(int &x, int &y)
{
    int  temp;
    temp = x; /* save the value of x */
    x = y; /* put y into x */
    y = temp; /* put x into y */
 }
```

```
#include <iostream>
using namespace std;
 // function declaration
void swap(int &x, int &y);
int main ()
{
   // local variable declaration:
   int a = 100;
   int b = 200;
   cout << "Before swap, value of a :" << a << endl;
   cout << "Before swap, value of b :" << b << endl;

   // calling a function to swap the values.
   swap(a, b);
   cout << "After swap, value of a :" << a << endl;
   cout << "After swap, value of b :" << b << endl;
   return 0;
}
```

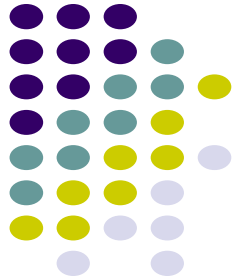


Functions in C++:

Default Values for Parameters :

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

int sum(int a, int b=20)
{
   int result;
   result = a + b;
   return (result);
 }
```

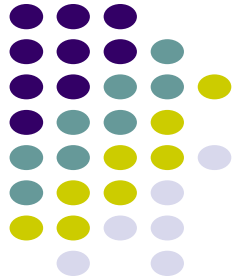
```
int main ()
 {
    // local variable declaration:
    int a = 100;
    int b = 200;
    int result;
    // calling a function to add the values.
    result = sum(a, b);
    cout << "Total value is :" << result << endl;
    // calling a function again as follows.
    result = sum(a);
    cout << "Total value is :" << result << endl;

    return 0;
 }
```



Introduction to Classes and Objects

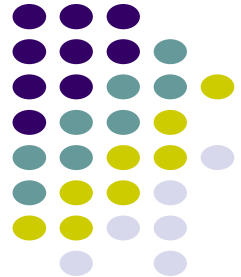
The classes are the most important feature of C++ that leads to Object Oriented programming. Class is a user defined data type, which holds its own data members and member functions, which can be accessed and used by creating instance of that class.

The variables inside class definition are called as data members and the functions are called member functions.



Introduction to Classes and Objects

1. Class name must start with an uppercase letter(Although this is not mandatory). If class name is made of more than one word, then first letter of each word must be in uppercase. *Example*, class Study, class StudyTonight etc
2. Classes contain, data members and member functions, and the access of these data members and variable depends on the access specifiers (discussed in next section).
3. Class's member functions can be defined inside the class definition or outside the class definition.
4. Class in C++ are similar to structures in C, the only difference being, class defaults to private access control, where as structure defaults to public.
5. All the features of OOPS, revolve around classes in C++. Inheritance, Encapsulation, Abstraction etc.
6. Objects of class holds separate copies of data members. We can create as many objects of a class as we need.
7. Classes do posses more characteristics, like we can create abstract classes, immutable classes, all this we will study later.



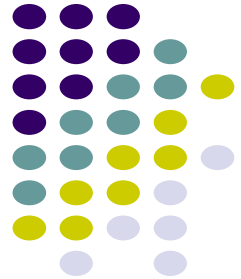
Introduction to Classes and Objects

Objects

Each object has different data variables. Objects are initialised using special class functions called **Constructors**. We will study about constructors later.

And whenever the object is out of its scope, another special class member function called **Destructor** is called, to release the memory reserved by the object. C++ doesn't have Automatic Garbage Collector like in JAVA, in C++ Destructor performs this task.

```
class Abc
{
 int x;
 void display(){} //empty function
};

in main()
{
 Abc obj;   // Object of class Abc created
}
```



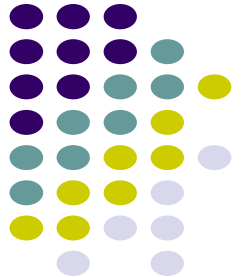
Access Control in Classes

Access specifiers in C++ class defines the access control rules. C++ has 3 new keywords introduced, namely,

1. public
2. private
3. protected

```
class ProtectedAccess
{
 protected:   // protected access
 int x;            // Data Member
 void display();   // Member Function
}
```

Protected, is the last access specifier, and it is similar to private, it makes class member inaccessible outside the class. But they can be accessed by any subclass of that class. (If class A is inherited by class B, then class B is subclass of class A.

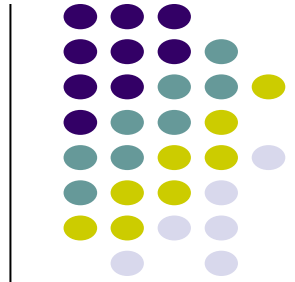


Defining Class and Declaring Objects

```
class ClassName
{
 Access specifier:
 Data members;
 Member Functions(){}
};
```

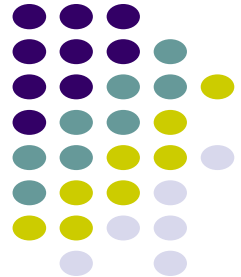
```
class Student
{
 public:
 int rollno;
 string name;
};
```

```
class Student
{
public:
int rollno;
string name;
}A,B;
```



Accessing Data Members of Class

```
class Student
{
 public:
 int rollno;
 string name;
};

int main()
{
 Student A;
 Student B;
 A.rollno=1;
 A.name="Adam";

 B.rollno=2;
 B.name="Bella";

 cout <<"Name and Roll no of A is :"<< A.name << A.rollno;
 cout <<"Name and Roll no of B is :"<< B.name << B.rollno;
}
```

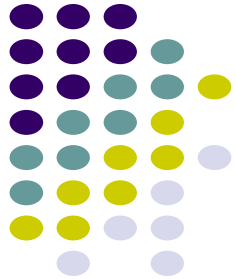


Accessing Data Members of Class

Accessing Private Data Members

```
class Student
{
 private:    // private data member
 int rollno;

 public:     // public accessor and mutator functions
 int getRollno()
 {
  return rollno;
 }

 void setRollno(int i)
 {
  rollno=i;
 }

};
```

```
int main()
{
 Student A;
 A.rollono=1;  //Compile time error
 cout<< A.rollno; //Compile time error

 A.setRollno(1);  //Rollno initialized to 1
 cout<< A.getRollno(); //Output will be 1
}
```



Constructors

Constructors are of three types :

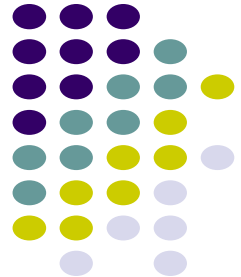
1. Default Constructor
2. Parametrized Constructor
3. Copy Constructor

```
class Cube
{
int side;
public:
Cube()
 {
  side=10;
 }
};

int main()
{
Cube c;
cout << c.side;
}                                   // Output : 10
```

Default

```
class Cube
{
 public:
 int side;
 Cube(int x)
  {
   side=x;
  }
};

int main()
{
 Cube c1(10);
 Cube c2(20);
 Cube c3(30);
 cout << c1.side;
 cout << c2.side;
 cout << c3.side;
}                                   // OUTPUT : 10 20 30
```

Parameterized



Constructors

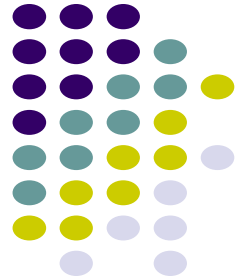
Copy Constructor

```
class Samplecopyconstructor
{
private:
int x, y; // data members
public:
Samplecopyconstructor(int x1, int y1)
{
x = x1;
y = y1;
}
// Copy constructor
Samplecopyconstructor (const Samplecopyconstructor &sam)
{
x = sam.x;
y = sam.y;
}
void display()
{
cout<<x<<" "<<y<<endl;
}
};
int main()
{
Samplecopyconstructor obj1(10, 15); // Normal constructor
Samplecopyconstructor obj2 = obj1; // Copy constructor
cout<<"Normal constructor : ";
obj1.display();
cout<<"Copy constructor : ";
obj2.display();
return 0;
}
```

Output:

Normal constructor : 10 15

Copy constructor : 10 15



Constructors

Constructor Overloading

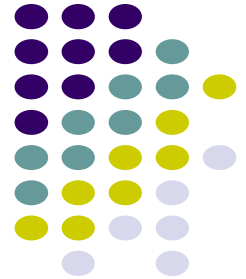
```
class Student
{
 int rollno;
 string name;
 public:
 Student(int x)
 {
  rollno=x;
  name="None";
 }
 Student(int x, string str)
 {
  rollno=x ;
  name=str ;
 }
};
```

```
int main()
{
 Student A(10);
 Student B(11,"Ram");
}
```

Note: Whenever we are using para.. constructor… without default constructor then we can declare like

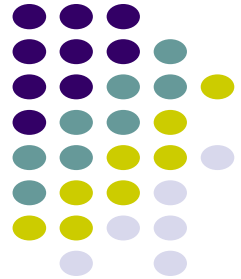
Student S;

it will lead to a compile time error, because we haven't defined default constructor, and compiler will not provide its default constructor because we have defined other parameterized constructors.



Destructors

```
class A
{
A()
 {
  cout << "Constructor called";
 }

~A()
 {
  cout << "Destructor called";
 }
};

int main()
{
 A obj1;   // Constructor Called
 int x=1;
 if(x)
  {
    A obj2;  // Constructor Called
  }   // Destructor Called for obj2
} //  Destructor called for obj1.
```



Types of Member Functions

1.Simple functions
2.Static functions
3.Const functions
4.Inline functions
5.Friend functions

Static Function

```
class X
{
 public:
 static void f(){};
};

int main()
{
 X::f();   // calling member function directly with
class name
}
```

These functions cannot access ordinary data members and member functions, but only static data members and static member functions.
It doesn't have any "this" keyword which is the reason it cannot access ordinary members

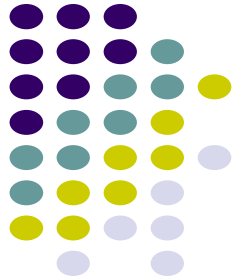
Const Member functions

When used with member function, such member functions can never modify the object or its related data members.

//Basic Syntax of const Member Function

```
void fun() const {}
```

Inline functions
All the member functions defined inside the class definition are by default declared as Inline



Types of Member Functions

Friend functions

Friend functions are actually not class member function. Friend functions are made to give **private** access to non-class functions.

You can declare a global function as friend, or a member function of other class as friend.

```
class WithFriend
{
 int i;
 public:
 friend void fun(); // Global function as friend
};
void fun()
{
 WithFriend wf;
 wf.i=10;  // Access to private data member
 cout << wf.i;
}
int main()
{
fun(); //Can be called directly
}
```

We can also make an entire class as friend class

```
class Other
{
 void fun();
};

class WithFriend
{
 private:
 int i;
 public:
 void getdata();  // Member function of class WithFriend
 friend void Other::fun();   // making function of class Other as friend here
 friend class Other;  // making the complete class as friend
};
```

When we make a class as friend, all its member functions automatically become friend functions.
Friend Functions is a reason, why C++ is not called as a pure Object Oriented language. Because it violates the concept of Encapsulation.