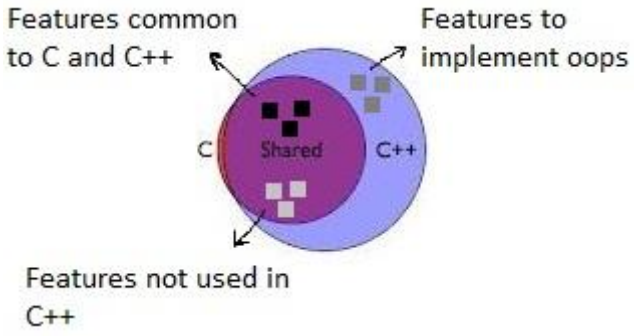
Agenda:

- How C++ differs from C
- Variable declaration
- Function overloading
- Reference parameters
- Operator overloading
- cin ,cout and Formatting
- Constant pointers
- Pointers to constant

How C++ differs from C



- C is procedural language whereas C++ is object oriented language.
- Most C Programs can be compiled in C++ compiler.
- Following are the other basic differences along with examples.

Variables Declaration

- Variables can declare anywhere in the program.
- **Declaration of Variables**: Variable to be declared before it use.

Data type: variable_list;

For example: int a,b;

string name;

float divide;

Example:

//Sum of two intergers	**//sum of integers upto given number**
#include<iostream>	#include<iostream>
using namespace std;	using namespace std;

```
void main()
{
   //declaring the variables
   int a,b;
   //initializing the variables
   a=5;
   b=6;
   int sum=0;
   sum=a+b;
   //Printing the sum
   cout << sum;
}
```

```
void main()
{
    int n;
    int sum=0;
    cout<<"Enter number, upto you want the sum
          "<<endl;
    cin>>n;    //gets the value
    for(int i=0;i<=n;i++)
    {
       sum+=i;   //adding the number
    }
    //prints the sum of all numbers
    cout<<"The sum of the digits:"<<sum;
    return 0;
}
```

Function overloading

Function overloading allows us to create several functions having same name which are different from each other in the type of input and output. (i.e. Functions having same name with different signatures.)

- The signature can differ by number of arguments and by different data types.
- It is usually used to enhance the readability of the program.
- Rules for function overloading
  - Overloaded function must differ from their data types.
  - The same function name is used for various instances of function call.

Example:

```
// Number of arguments different
#include<iostream>
using namespace std;
s
int sum(int x,int y)
{
    cout<<x+y;
}
int sum(int x,int y,int z)
```

```
{
    cout<<x+y+z;
}
int sum(double x,double y)
{
    cout<<x+y;
}
//main function
int main()
{
    //calling the function
    sum(10,20);
    sum(10,20,30);
    sum(10.20,30.7689);
    return 0;
}
```

Optional Parameters

- Parameters of a function can be given some default value.
- At the time of calling the function if the argument is not provided, default value will be used for the parameter but if the argument is provided the parameter will be initialized with the value of the argument.
- Once a parameter is provided with default value, it is compulsory that all subsequent parameters also must be having default value.

Example:

```
sum(int a,int b=0);
sum(int a,int b=0,int c);  //this is incorrect
sum(int a,int b,int c=0,int=2); //this is correct
```

Reference Variables

- Reference variables is the name given to an existing storage i.e., alias name to store the address of another variable or object or function.
- Reference variables can used as parameters are called as reference parameters.
- References are mainly used in functions where we want to return multiple values from the function.
- Passing the values as arguments into function is called pass by value a copy of arguments is passed to function.

Example:

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

void Foo(int a,int &b, int *pA)
{
    a++;
    b++;
    *pA = *pA + 1;
}
void swap(int &a,int &b)
{
    int tmp;
    tmp = a;
    a = b;
    b = tmp;
}

int main()
{
    int n;s
    n = 10;
    int &m = n;     /*Storing n value in the address of m*/
    n++;
    cout << m << " " << n << endl;    /*printing m,n values*/
    m++;
    cout << m << " " << n << endl;    /*printing the m,n values after incrementing the m value*/
    int x,y,z;
    x = y = z = 10;
```

```
    Foo(x,y,&z);    /*calling Foo() function from main() by passing x,y values and address of z as paarmeters*/
    cout << x << " " << y << " " << z << endl;
    x = 100;
    y = 200;
    swap(x,y);
    cout << x << " " << y << endl;
}
```

Operator overloading

- Different operators having different implementations according to supplied operands.
- Operator overloading redefines the meaning of operator in the program.
- These are mainly used in user-defined data types like objects, structures.
- The standard behavior of operators for built-in (primitive) types cannot be changed by overloading, that is, you can't overload operator+(int,int).
- Some of the operators does not support overloaded namely,
    - "."(Member access or dot operator),
    - "?:"(Ternary or conditional operator),
    - "::"(Scope Resolution operator),
    - ".*"(pointer to member operator),
    - "sizeof"(object size operator)
    - "typeid"(object type operator). .

Example:

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

struct Complex
{
    int Real;
    int Imaginary;
};

Complex operator+(const Complex &c1,const Complex &c2)
{
    Complex c3;
    c3.Real = c1.Real + c2.Real;
```

```
    c3.Imaginary = c1.Imaginary + c2.Imaginary;
    return c3;
}
Complex& operator+(const Complex &c,int n)
{
    Complex c3;
    c3.Real = c.Real + n;
    c3.Imaginary = c.Imaginary;
    return c3;
}
Complex& operator+(int n, const Complex &c)
{
    return c+n;
}
ostream& operator<<(ostream &out,Complex c)
{
    out << c.Real << " " << c.Imaginary << endl;
    return out;
}

int main()
{
    Complex c1, c2;
    c1.Real = 10;
    c1.Imaginary = 100;
    c2.Real = 20;
    c2.Imaginary = 200;
    Complex c3 = c1 + c2;
    cout << c3.Real << " " << c3.Imaginary << endl;  //display real,imaginary using cout without operator overloading
    c3 = c1 + 10;  //By using '+' operator by passing structure and number
    c3 = 10 + c1;  //Here,we are calling '+' opeartor  by number and structure
    cout << c1 << c2 << c3;  //By using operator overloading displaying complex strcture
}
```


Basics of Console Input and Output

- cin and cout are used for console input and output streams respectively.

- cin andcout are equivalents of scanf and printf in C language.
- cin is an object of type istream and cout is an object of type ostream. Both are byte streams
- cin is used for reading the data from keyboard into program.
- cout is used for writing the data to monitor.
- cin.ignore() – to extract and discard data from the stream

Example:

To Print a Table of a Number:

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

int main()
{
	int number;
	cout << "Please enter the number: ";
	cin >> number;
	for(int i=1;i<=10;i++)
	{
		cout << number << " * " << i << " = " << number*i << endl;  //Multiplying the number with  i  value
	}
	cout << "Enter the two numbers to be added: ";
	int n1,n2;
	cin >> n1 >> n2;
	cout << "Sum of two numbers is: " << n1+n2 << endl;  //Printing sum of two numbers
}
```

String Inout and Output:

#include <string>

string is a built-in class can be used as a datatype in C++;

string str;

cin >> str; - This can read a string until it finds a space. Every thing after first space character is ignored.

getline(cin,str); - This function reads complete string including spaces till it finds a new line character and stores the same in str variable.

getline(cin,str,'#'); - Reads everything including new line till it finds '#'

```
#include <iostream>
#include <string>
```

```
using namespace std;


int main()
{
    string firstName, lastName;
    cout << "What is your first name: ";
    cin >> firstName;
    cout << "What is your last name: ";
    cin >> lastName;
    cout << "Hello " << firstName << " " << lastName << endl;//Printing firstname and lastname
    cout << "What is your full name: ";
    cin.ignore();//statement will discard the input value
    string fullName;
    getline(cin,fullName);//reads complete string including spaces,everything aftre first space is ignored
    cout << fullName;
    return 0;
}
```

Constant Pointers

- Constant pointers are those whose address cannot be modified. Once a constant pointer is pointed to a variable then it cannot point to any other variable.
- **Syntax:** PointerType *const PointerName;

Example:

```
#include<iostream>
using namespace std;
int main()
{
   char *const str="TAT";
   str ++;          //Error:deprecated conversion rom string to char*     cout<<str;
   return 0;
}
```

Program output:

Error: Cannot modify a const object in function main()

Pointer to a Constant

- In pointer to a constant, we can change the value by using the actual variable which was not possible with pointers.

Example:

```
#include<iostream>
using namespace std;
int main()
{
   int k = 20;
   int const *p = &k;
   k=30;
   cout<<*p;              //30
}
```

Dynamic Memory Allocation

In C, there's only one major memory allocation function: malloc. You use it to allocate both single elements and arrays:

```
int *x = malloc( sizeof(int) );
int *x_array = malloc( sizeof(int) * 10 );
```

and you always release the memory in the same way:

```
free( x );
free( x_array );
```

In C++, however, memory allocation for arrays is somewhat different than for single objects; you use the new[] operator, and you must match calls to new[] with calls to delete[] (rather than to delete).

```
int *x = new int;
int *x_array = new int[10];
delete x;
delete[] x_array;
```

The short explanation is that when you have arrays of objects, delete[] will properly call the destructor for each element of the array, whereas delete will not.