Preview

- More Recursion Examples
  - Sum of Range
  - The Fibonacci Series
  - Greatest Common Devisor
  - Binary Search
- Operator Overloading
  - As a member function
  - As a Friend function

More Recursive Examples

Sum of a Range of Array Elements with Recursion

```
int [] A ={1, 2, 3, 4, 5, 6, 7, 8, 9}
int sum;
sum = rangeSum (A, 3, 7);
// it returns  A[4]+A[5]+A[6]+A[7]+ A[8]
```


More Recursive Examples

```
#include <iostream>
using namespace std;
// A Function Prototype
int rangeSum(int[], int, int);

int main()
{
        int numbers[] = {1, 2, 3, 4, 5, 6, 7, 8, 9};
        cout <<"The sum of elements 2 through 5 is ";
        cout << rangeSum(numbers, 2, 5)<<endl;
        return 0;
}

// simple function without recursion
int rangeSum(int A[], int start, int end)
{
    int rgsum =0;
        for (int i = start; i <=end; i++)
                rgsum += A[i];
        return rgsum;
}
```


More Recursive Examples

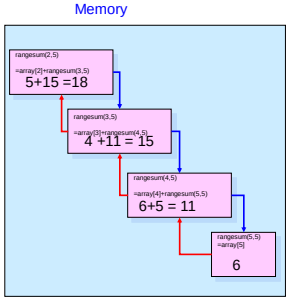
```
#include <iostream>
using namespace std;
// A Function Prototype
int rangeSum(int[], int, int);

int main()
{
        int numbers[] = {1, 2, 3, 4, 5, 6, 7, 8, 9};
        cout <<"The sum of elements 2 through 5 is ";
        cout << rangeSum(numbers, 2, 5)<<endl;
        return 0;
}

// simple function with recursion
int rangeSum(int A[], int start, int end)
{
      if (start > end)
         return 0;
      else
         return A[start] + rangeSum(A, start + 1, end);
}
```


More Recursive Examples

rangesum(2, 5) = A[2] + rangesum(3, 5)
rangesum(3, 5) =A [3]+ rangesum(4, 5)
rangesum(4,5) = A[4] + rangesum(5,5)
rangesum(5,5) = array[5]



More Recursive Examples

- The Fibonacci Series
  1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89,…..

$$f(n) = \begin{cases} 1 & n = 0 \text{ or } n = 1 \\ f(n\text{-}1) + f(n\text{-}2) & n > 1 \end{cases}$$

More Recursive Examples

```
#include <iostream>
using namespace std;
// A Function Prototype
int fib(int n);

int main()
{
          cout <<"The first ten numbers in the Fibonacci series are: ";

          for (int i = 0; i < 10; i++)
                    cout << fib(i) << " ";
          cout<< endl;
          return 0;
}
// simple function without recursion
int fib (int x)
{
          int f2=1, f1=1, newTerm=0;
          if (x==0 || x==1)
                    return 1;
          for (int i=2; i<=x; i++)
          {
                    newTerm = f1 + f2;
                    f2 = f1;
                    f1 = newTerm;
          }
          return newTerm;
}
```


More Recursive Examples

```
#include <iostream>
using namespace std;
// A Function Prototype
int fib(int n);


int main()
{
          cout <<"The first ten numbers in the Fibonacci series are: ";

          for (int i = 0; i < 10; i++)
                    cout << fib(i) << " ";
          cout<< endl;
          return 0;
}

// simple function with recursion
int fib(int n)
{
      if (n == 0 || n == 1)
         return 1;
      else
         return fib(n - 1) + fib(n - 2);
}
```


More Recursive Examples

- Finding the Greatest Common Devisor

$$GCD(x,y) = \begin{cases} y & \text{if y divides x evenly} \\ \text{GCD(y, remainder of x/y)} & \text{otherwise} \end{cases}$$

More Recursive Examples

```
#include <iostream>
using namespace std;
// A Function Prototype
int GCD (int, int);

int main()
{
          int x, y;
          cout << "two integer to finde  Greatest Common Devisor: ";
          cin >> x >> y;

          cout << "GCD (" << x <<", " << y <<" ) = ";
          cout<< GCD (x, y) <<endl;
          return 0;
}

//Non-Recursive versfor Greatest Common Devisor
int GCD(int a, int b)
{
    while( 1 )
    {
        a = a % b;
        if( a == 0 )
            return b;
        b = b % a;
        if( b == 0 )
            return a;
    }
}
```


More Recursive Examples

```
#include <iostream>
using namespace std;
// A Function Prototype
int GCD (int, int);


int main()
{
          int x, y;
          cout << "two integer to finde  Greatest Common Devisor: ";
          cin >> x >> y;

          cout << "GCD (" << x <<", " << y <<" ) = ";
          cout<< GCD (x, y) <<endl;
          return 0;
}

//Recursive function for Greatest Common Devisor
int GCD(int x, int y)
{
          if (x % y == 0)
                    return y;
          else
                    return GCD(y, x % y);
}
```


More Recursive Examples

- Define a Boolean function str_length() which returns the length of c-string

```
//Non-Recursive version of strLength
int strLength (const char A[])
{
        int i=0;
        while (A[i]!='\0')
            i++;
        return i;
}
```

More Recursive Examples

- Define a Boolean function str_length() which returns the length of c-string

```
//Recursive version of Str_Cmp

int StrLength(char A[], int first)
{
       if (A[first]!= '\0')
              return 0;

       else
              return 1+ strLength(A, first+1);
}
```


Operator Overloading

- Operator overloading is used to give special meaning to the commonly used operators (such as +, -, * etc.) with respect to a class.
- By overloading operators, we can control or define how an operator should operate on data with respect to a class.
- Operators are overloaded in c++ by creating operator functions either as a member or as a friend of a class.

Operator Overloading as a Member

```
//MyClass.h
#ifndef MYCLASS_H
#define MYCLASS_H

class MyClass
{
private:
       int sub1, sub2;
public:
       // constructor with default arguments
       MyClass (int=0, int =0);
       // notice the declaration
       MyClass operator +(MyClass);
       MyClass operator -(MyClass);
       bool operator == (MyClass);
       void Display();
  };

#endif
```


Operator Overloading as a Member

```
// MyClass.cpp
// Member functions for MyClass class

#include <iostream>
#include "MyClass.h"
using namespace std;


//constructure
MyClass::MyClass(int x, int y)
{
         sub1=x;
         sub2=y;
}

// overloaded operator - for MyClass object
MyClass MyClass::operator -(MyClass ob)
{
    MyClass temp;

    // add the data of the object that generated the call
    // with the data of the object passed to it and store in temp
    temp.sub1=sub1 - ob.sub1;
    temp.sub2=sub2 - ob.sub2;
    return temp;
}
```


Operator Overloading as a Member

```
MyClass MyClass::operator +(MyClass ob)
{
         MyClass temp;

    // add the data of the object that generated the call
    // with the data of the object passed to it and store in temp
    temp.sub1=sub1 + ob.sub1;
    temp.sub2=sub2 + ob.sub2;
    return temp;
}

// overloaded operator == for MyClass object
bool MyClass::operator ==(MyClass ob)
{
         if (sub1 == ob.sub1 && sub2== ob.sub2)
                  return true;
         else
                  return false;
}

//display two private members
void MyClass::Display()
{
         cout <<"sub1 = "<<sub1 << endl;
         cout <<"sub2 = "<<sub2 << endl;
}
```


Operator Overloading as a Member

```
// Overator overloading example
#include <iostream>
#include "MyClass.h"
using namespace std;
void main()
  {
    MyClass ob1(10,90);
    MyClass ob2(90,10);
    // this is valid
    ob1=ob1+ob2;
    ob1.Display();

         ob2 =ob1-ob2;
         ob2.Display();

         MyClass ob3 (10, 10);
         MyClass ob4(10,10);
         if (ob3 == ob4)
                  cout << "Two objects have same values"<<endl;
         else
                  cout <<"Two objects have different values"<<endl;
  }
```

Operator Overloading as a Friend

- If an overloaded operator defined as a friend function of a class, it can be used in other class as a friend.
- If an overloaded operator defined as a friend function of a class, we need pass one more object as a parameter, since a friend function is not a member of a class

Operator Overloading as a Friend

```
// DateClass.h for a overloading operator example as freinds
#include <iostream>
using namespace std;
#ifndef DATECLASS_H
#define DATECLASS_H
class Date{
        int month;
        int day;
        int year;
public:
        Date(int =0, int =0, int =0);
        friend ostream& operator <<(ostream& stream, Date );
        friend istream& operator >>(istream& stream, Date *);
        friend bool operator == (Date, Date);
        friend bool operator != (Date, Date);
};

#endif
```


Operator Overloading as a Friend

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

// Constructor
Date::Date(int month, int day, int year)
{
          month = month;
          day = day;
          year =year;
}

//overloaded operator for output
ostream& operator <<(ostream& stream, Date date)
{
          stream << date.month<<'/'<<date.day<<'/'<<date.year<<endl;
          return(stream);
}
//overloaded operator for input
istream& operator >>(istream& stream, Date *date)
{
          stream >> date->month;
          stream >> date->day;
          stream >> date->year;
          return(stream);
}
```


Operator Overloading as a Friend

```
//overloaded overator for ==
bool operator == (Date one, Date two)
{
        if (one.day != two.day)
                return false;
        else if (one.month != two.month)
                return false;
        else if (one.year != two.year)
                return false;
        return true;
}
//overloaded operator for !=
bool operator !=(Date one, Date two)
{
        if (one.day != two.day)
                return true;
        else if (one.month != two.month)
                return true;
        else if (one.year != two.year)
                return true;
        return false;
}
```


Operator Overloading as a Friend

```
// Operator overloading example as a friend function
#include <iostream>
#include "DateClass.h"
using namespace std;
void main()
{
          Date HerBirthday;
          Date YourBirthday;
          cout << "When is her birthday? (mm dd yy) ";
          cin >> &HerBirthday;
          cout << "When is your birthday? (mm dd yy) ";
          cin >> &YourBirthday;
          cout << "Your birthday is " << YourBirthday;
          cout <<"Her birthday is "<< HerBirthday;
          if (HerBirthday == YourBirthday)
             cout << " Your and Her birthday are the same day! " << YourBirthday;
          else if (HerBirthday != YourBirthday)
             cout << " Your and Her birthday are Different day! "<<endl;

}
```