```
/*                      apointers1.cpp
 *
 *   Synopsis  -  Accepts a line of text as input from standard
 *                input. Parses the input text to find individual
 *                words, counts them, displays the count, and
 *                displays the words in reverse order.
 *
 *   Objective - Illustrates use of an array of pointers to char.
 */

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


int main()
{
        char instring[80];
        char *words[50],
            *current;
        int  i = 1;

        cout <<  "Enter text with words delimited by blanks: " << endl ;
        cin.getline(instring,80);

        words[0] = current = instring;

        while ( ( current = strchr( current, ' ' )) != NULL ) {
                *current++ = '\0';
                words[i++] = current;
        }

        cout << "There were " << i << " words in that line " << endl;
        cout << "In reverse order they are : " << endl;
        for ( --i; i >= 0; i-- )
                cout << words[i] << endl;
}
output
```

```
Enter text with words delimited by blanks:
the rain in spain falls mainly in the plain
There were 9 words in that line
In reverse order they are :
plain
the
in
mainly
falls
spain
in
rain
the
```

```
// apointers2.cpp
// double indirection
#include <iostream>
using namespace std;
int main()
{
        int myInt=3;
        int * myIntPtr=&myInt;
        int ** myIntPtrPtr=&myIntPtr;
        cout << myInt << endl;
        cout << *myIntPtr <<  endl;
        cout << &myInt <<   endl;
        cout << myIntPtr << endl;
        cout << &myIntPtr << endl;
        cout << myIntPtrPtr <<  endl;
        cout << **myIntPtrPtr << endl;
        cout << &myIntPtrPtr << endl;
}
```

```
output
3
3
0x324723e8
0x324723e8
0x324723e4
0x324723e4
3
```

0x324723e0

```
/*              apointers3.cpp
 *
 *   Synopsis  -  Outputs information, strings, and individual
 *                characters in an array of pointers to type char.
 *
 *   Objective -  To illustrate double indirection and use the
 *                pointer to a pointer to traverse an array of
 *                pointers.
 */


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


int main()
{
        char *ptrarray[] = { "George",
                             "Elliot's",
                                 "Oldest",
                                 "Girl",
                                 "Rode",
                                 "A",
                                 "Pig",
                                 "Home",
                                 "Yesterday",
                                 ""
                               };

        char **ptrptr = ptrarray;

        cout <<  "sizeof( ptrarray )   sizeof( ptrptr )  " <<
                                 sizeof( ptrarray ) << " " <<  sizeof( ptrptr ) << endl;
        cout <<  "ptrarray    ptrptr  "  <<
                                 ptrarray << " " << ptrptr << endl;
        cout <<  "ptrarray[0]  *ptrptr " <<
                                 ptrarray[0] << " " << *ptrptr << endl;
```

```
cout <<  "ptrarray[0]  *ptrptr  "  <<
                        ptrarray[0] << " " << *ptrptr << endl;
cout <<  "ptrarray[1]   *( ptrptr+1 ) " <<
                        ptrarray[1] << " " << *( ptrptr+1 ) << endl;

cout <<  "*ptrarray[0]   **ptrarray "  <<
                        *ptrarray[0] << " " << **ptrarray  << endl;
cout <<  "ptrarray[0][4]  *(*ptrarray + 4) " <<
                        ptrarray[0][4] << " " << *( *ptrarray+4 ) << endl;

for (  ; strcmp( *ptrptr, ""); ptrptr++ )
        cout <<   *ptrptr << " " ;
cout <<  endl;

for ( ptrptr = ptrarray; strcmp( *ptrptr, "" ); ptrptr++ )
        cout <<   **ptrptr ;
cout <<  endl;


ptrptr = ptrarray;
cout << "ptrptr " << ptrptr << endl  ;
ptrptr = ptrarray;
cout << "b:*ptrptr  " << *ptrptr << endl  ;
ptrptr = ptrarray;
cout << "c:**ptrptr " <<**ptrptr << endl  ;
ptrptr = ptrarray;
cout << "d: ptrptr+1 " << ptrptr+1 << endl  ;
ptrptr = ptrarray;
cout << "e: *(ptrptr+1)  " << *(ptrptr+1) << endl  ;
ptrptr = ptrarray;
cout << "f: **(ptrptr+1) " <<**(ptrptr+1) << endl  ;
ptrptr = ptrarray;
cout << "g: *(*(ptrptr+1)+2) " <<*(*(ptrptr+1)+2) << endl  ;
ptrptr = ptrarray;
cout << "h: **ptrptr+1 " <<**ptrptr+1 << endl  ;
ptrptr = ptrarray;
cout << "i: *ptrptr+1  " << *ptrptr+1 << endl  ;
ptrptr = ptrarray;
cout << "j: *ptrptr[1] " << *ptrptr[1] << endl  ;
```

```
        ptrptr = ptrarray;
        cout << "k: *(*ptrptr+2) " << *(*ptrptr+2) << endl  ;
        ptrptr = ptrarray;
        cout << "g2: *(*(ptrptr+1)+4) " <<*(*(ptrptr+1)+4) << endl  ;
        ptrptr = ptrarray;
        cout << "j2: *ptrptr[6] " << *ptrptr[6] << endl  ;
        ptrptr = ptrarray;
        cout << "k2: *(*ptrptr+3) " << *(*ptrptr+3) << endl  ;

}
output
sizeof( ptrarray )   sizeof( ptrptr )  40 4
ptrarray    ptrptr  0x4b2f2558 0x4b2f2558
ptrarray[0]  *ptrptr George George
ptrarray[0]  *ptrptr  George George
ptrarray[1]   *( ptrptr+1 ) Elliot's Elliot's
*ptrarray[0]   **ptrarray G G
ptrarray[0][4]  *(*ptrarray + 4) g g
George Elliot's Oldest Girl Rode A Pig Home Yesterday
GEOGRAPHY
ptrptr 0x4b2f2558
b:*ptrptr  George
c:**ptrptr G
d: ptrptr+1 0x4b2f255c
e: *(ptrptr+1)  Elliot's
f: **(ptrptr+1) E
g: *(*(ptrptr+1)+2) l
h: **ptrptr+1 72
i: *ptrptr+1  eorge
j: *ptrptr[1] E
k: *(*ptrptr+2) o
g2: *(*(ptrptr+1)+4) o
j2: *ptrptr[6] P
k2: *(*ptrptr+3) r
```

```
/*              command1.cpp
 *   Synopsis  -  Prints the value of argc and the command
 *                line arguments.
 *
 *   Objective -  To illustrate how command line arguments work
 *                and to demonstrate two techniques for accessing
 *                the arguments. Pointer notation used.
 */

#include <iostream>
using namespace std;

int main( int argc, char **argv )                  // pointer notation
{
        int index ;

        cout << "There were " << argc << " arguments on the command line
"<< endl;

        cout << "They are " << endl;

        for ( index = 0; index < argc; index++ )
                cout << argv[index] << " " << endl;

        while ( argc-- > 0 )
                cout << *argv++ << " ";
}
output
There were 4 arguments on the command line
They are
C:\JERRYL~1\CPP\EXAMPLES\COMMAND1.EXE
abc
def
ghi
C:\JERRYL~1\CPP\EXAMPLES\COMMAND1.EXE abc def ghi

riend1
#include <iostream>

using namespace std;
```

```
class rectangleType
{
    friend void rectangleFriend(rectangleType recObject);
public:
    void setDimension(double l, double w);
      //Function to set the length and width of the rectangle.
      //Postcondition: length = l; width = w;

    double getLength() const;
      //Function to return the length of the rectangle.
      //Postcondition: The value of length is returned.

    double getWidth() const;
      //Function to return the width of the rectangle.
      //Postcondition: The value of width is returned.

    double area() const;
      //Function to return the area of the rectangle.
      //Postcondition: The area of the rectangle is
      //               calculated and returned.

    double perimeter() const;
      //Function to return the perimeter of the rectangle.
      //Postcondition: The perimeter of the rectangle is
      //               calculated and returned.

    void print() const;
      //Function to output the length and width of
      //the rectangle.

    rectangleType();
      //Default constructor
      //Postcondition: length = 0; width = 0;

    rectangleType(double l, double w);
      //Constructor with parameters
      //Postcondition: length = l; width = w;

private:
    double length;
```

```
    double width;
};




void rectangleFriend(rectangleType recFriendObject)
{
    cout << "recFriendObject area: " << recFriendObject.area()
         << endl;

    recFriendObject.length = recFriendObject.length + 5;
    recFriendObject.width = recFriendObject.width + 5;

    cout << "After increasing length and width by 5 units "
         << "each, \n      recFriendObject area: "
         << recFriendObject.area() << endl;
}

void rectangleType::setDimension(double l, double w)
{
    if (l >= 0)
        length = l;
    else
        length = 0;

    if (w >= 0)
        width = w;
    else
        width = 0;
}

double rectangleType::getLength() const
{
    return length;
}

double rectangleType::getWidth()const
{
```

```
    return width;
}

double rectangleType::area() const
{
    return length * width;
}

double rectangleType::perimeter() const
{
     return 2 * (length + width);
}

void rectangleType::print() const
{
    cout << "Length = "  << length
         << "; Width = " << width;
}

rectangleType::rectangleType(double l, double w)
{
    setDimension(l, w);
}

rectangleType::rectangleType()
{
    length = 0;
    width = 0;
}


//Friend Function Illustration


#include <iomanip>                          //Line 2


int main()                                  //Line 5
{                                          //Line 6
    rectangleType myYard(25, 18);                //Line 7
```

```
    cout << fixed << showpoint << setprecision(2); //Line 8

    cout << "myYard area: " << myYard.area()
         << endl;                              //Line 9

    cout << "Passing object myYard to the friend "
         << "function rectangleFriend." << endl;   //Line 10

    rectangleFriend(myYard);                       //Line 11

    return 0;                                  //Line 12
}                                             //Line 13
```

Output

```
myYard area: 450.00
Passing object myYard to the friend function rectangleFriend.
recFriendObject area: 450.00
After increasing length and width by 5 units each,
      recFriendObject area: 690.00
                                        //Line 13
```

```
// friend2.h
// friend functions
#include <iostream>
using namespace std;


class beta;             // needed for friendFunction declaration

class alpha
  {
  private:
     int data;
  public:
               alpha();
               friend int friendFunction(alpha, beta);  // friend function
  };
```

```
class beta
   {
   private:
      int data;
   public:
                beta() ;            // no-arg constructor
                friend int  friendFunction(alpha, beta);  // friend function
        };
// friend2i.cpp
// friend functions
#include <iostream>
using namespace std;
#include "friend2.h"

        alpha::alpha()  // no-arg constructor
        {
                data=3;
        }

        beta::beta()      // no-arg constructor
        {
                data=7;
        }

        int friendFunction(alpha a, beta b)           // function definition
        {
        return( a.data + b.data );
        }
// friend2.cpp
// friend functions
#include <iostream>
using namespace std;
#include "friend2.h"

   int main()
        {
        alpha aa;
        beta bb;
        cout <<  friendFunction(aa, bb);            // call the function
```

```
        }
output
10
// friend3
#include <iostream>
using namespace std;

class Foo; // Forward declaration of class Foo in order for example to
compile.
class Bar {
  private:
     int a;
  public:
     Bar(): a(0) {}
     void show(Bar& x, Foo& y);
     friend void show(Bar& x, Foo& y); // declaration of global friend
};

class Foo {
  private:
     int b;
  public:
     Foo(): b(6) {}
     friend void  show(Bar& x, Foo& y); // declaration of global friend
     friend void Bar::show(Bar& x, Foo& y); // declaration of friend from
other class
};

// Definition of a member function of Bar; this member is a friend of Foo
void Bar::show(Bar& x, Foo& y) {
  cout << "Show via function member of Bar" << endl;
  cout << "Bar::a = " << x.a << endl;
  cout << "Foo::b = " << y.b << endl;
}

// Friend for Bar and Foo, definition of global function
void show(Bar& x, Foo& y) {
  cout << "Show via global function" << endl;
  cout << "Bar::a = " << x.a << endl;
  cout << "Foo::b = " << y.b << endl;
```

```
}

int main() {
   Bar a;
   Foo b;

   show(a,b);
   a.show(a,b);
}
```

Output

```
Show via global function
Bar::a = 0
Foo::b = 6
Show via function member of Bar
Bar::a = 0
Foo::b = 6
```

Overloadable operators

```
 + - * / % ^ & | ~ ! ' = < > <+ >= ++ -- << >> == != && ||
 += -= /= %= ^= &= |= *= <<= >>= [] () -> ->* new delete
```

```
// this1.cpp
// the this pointer
#include <iostream>
using namespace std;

class where
        {

        public:
                void reveal();

        private:
                char charray[10];   // occupies 10 bytes
        };
```

```
void where::reveal()
{
 cout << "\nMy object's address is " << this;
}

int main()
{
where w1, w2, w3;   // make three objects
w1.reveal();        // see where they are
w2.reveal();
w3.reveal();
}
```

output

```
My object's address is 0x5cef23fa
My object's address is 0x5cef23f0
My object's address is 0x5cef23e6
```

//overload1.h

```
#ifndef H_OpOverClas
#define H_OpOverClass

class OpOverClass
{
public:
   void print() const;

                //Overload the arithmetic operators
   OpOverClass operator+(const OpOverClass&) const;
   OpOverClass operator*(const OpOverClass&) const;

   OpOverClass();
   OpOverClass(int i);

private:
   int a;
};
#endif
```

//overload1i.cpp

```
#include <iostream>
```

```
#include "overload1.h"

using namespace std;

void OpOverClass::print() const
{
	cout<<"The value of a is "<<a ;
}

OpOverClass::OpOverClass()
{
	a = 0;

}
OpOverClass::OpOverClass(int i)
{
	a = i;

}

OpOverClass OpOverClass::operator+
				(const OpOverClass& rightOperand) const
{
	OpOverClass temp;

	temp.a = a + rightOperand.a;

	return temp;
}

OpOverClass OpOverClass::operator*
				 (const OpOverClass& rightOperand) const
{
	OpOverClass temp;

	temp.a = a * rightOperand.a;

	return temp;
}
//overload1.cpp
```

```
#include <iostream>

#include "overload1.h"

using namespace std;

int main()
{
        OpOverClass u(23);              //Line 1
        OpOverClass v(10);              //Line 2
        OpOverClass w1;                                 //Line 3
        OpOverClass w2;                                 //Line 4

        cout<<"Line 5: u = ";           //Line 5
        u.print();                              //Line 6; output u
        cout<<endl;                                     //Line 7

        cout<<"Line 8: v = ";           //Line 8
        v.print();                              //Line 9; output v
        cout<<endl;                                     //Line 10

        w1 = u + v;                             //Line 11; add u and v

        cout<<"Line 12: w1 = ";                 //Line 12
        w1.print();                             //Line 13; output w1
        cout<<endl;                                     //Line 14

        w2 = u * v;                             //Line 15; multiply u and v

        cout<<"Line 16: w2 = ";                 //Line 16
        w2.print();                             //Line 17; output w2
        cout<<endl;                                     //Line 18

        return 0;
}
output
Line 5: u = The value of a is 23
Line 8: v = The value of a is 10
Line 12: w1 = The value of a is 33
Line 16: w2 = The value of a is 230
```

//overload2

// overload2.cpp

```
#include<iostream>
#include<stdio.h>

using namespace std;

#include <iostream>
using namespace std;

// Definition for date class
class Date
{
      public:
            Date();
            Date(int mon, int da, int yr);
            void display() const;

// overload the << and >>
             friend istream& operator >> (istream& input, Date& inputDate);
             friend ostream& operator << (ostream& output, Date&
outputDate);

// Member functions to overload operators
                  bool operator<(const Date&);   // less than
                  bool operator>(const Date&);   // greater than
                  bool operator==(const Date&);  // equal
                  Date operator+(int); // overloaded +  to add a day
                  Date operator+=(int);
                  Date operator++();  //prefix ++ operator
                  Date operator++(int);  //postfix ++ operator
                  Date& operator=(const Date&);  //assignment operator
                  Date (const Date& dateObject); // copy constructor

 private:
      int month;
      int day;
      int year;
} ;

int daysPerMonth[]={31,28,31,30,31,30,31,31,30,31,30,31};
//
// Definition for date class
//
//    int month;
//    int day;
//    int year;
```

```
Date::Date()
	{
		month=0;
		day=0;
		year=0;
		cout << "default constructor called " << endl;
	 }

Date::Date(int mon,int da, int yr)
	{
		month=mon;
		day=da;
		year=yr;
		cout << "non-default constructor called " << endl;
	}

Date::Date (const Date& dateObject) // copy constructor
{
		month=dateObject.month;
		day=dateObject.day;
		year=dateObject.year;
		cout << "copy constructor called " << endl;
	}

void Date::display() const
  {
		cout << month << '/' << day << '/' << year ;
  }

  // Date class function to overload < operator
bool Date::operator<(const Date &compareDate)
{
	if (year == compareDate.year)
	{
		if (month == compareDate.month)
			return day < compareDate.day;
		return month < compareDate.month;
	}
	return year < compareDate.year ;
}

bool Date::operator>(const Date &compareDate)
{
	if (year == compareDate.year)   // note this-> is not needed
	{
		if (month == compareDate.month)
			return day > compareDate.day;
		return month > compareDate.month;
	}
```

```
      return year > compareDate.year ;
}

// Date class function to overload == (equivalence) operator
bool Date::operator==(const Date &compareDate)
{
      return (year == compareDate.year &&
                  month == compareDate.month &&
                  day == compareDate.day);
}

// Date class function to overload +  operator
Date Date::operator+(int numberOfDays)
{
      Date addDate = *this;
      numberOfDays += addDate.day;
            while (numberOfDays> daysPerMonth[addDate.month-1]) // check for
a new month
        {
            numberOfDays -= daysPerMonth[addDate.month-1];
            if (++addDate.month == 13) // check to see if a new year
                  {
                        addDate.month=1;
                        addDate.year++;
                  }
            }
            addDate.day=numberOfDays;
            return addDate;
}

Date Date::operator+=(int numberOfDays)
{
      *this = *this + numberOfDays;   // class arithmetic
      return *this;
}

// overloaded prefix operator
Date Date::operator++()
{
      *this = *this + 1;        // class arithmetic
      return *this;
}

// overloaded postfix operator
Date Date::operator++(int)
{
      Date addOne= *this;
      *this=*this+1;         // class arithmetic
      return addOne;
}
```

```
	ostream& operator << (ostream& output, Date& outputDate)  // cout
operator
	{
	output << outputDate.month << "/" << outputDate.day << "/"
		<< outputDate.year << endl;
	return output;
	}

	istream& operator >> (istream& input, Date& inputDate)  // cin
operator
	{
	cout << "\nEnter month: ";
	input >> inputDate.month;
	cout << "\nEnter day: ";
	input >> inputDate.day;
	cout << "\nEnter year: ";
	input >> inputDate.year;
	return input;
	}

// Date class function to overload = (assignment) operator
// This function will override the default
Date&  Date::operator=(const Date &assignDate)
{
	month = assignDate.month;
	day = assignDate.day;
	year = assignDate.year;
	cout << "copy assign operator invoked" << endl;
	return *this ;
}
// overload2.cpp
// illustrates overloading of operators for the date class
// test file

// test the overloading of the date class
int main()
{
  Date firstDate(3,6,2020);
  Date secondDate(2,6,2019);
  Date thirdDate(3,5,2020);
  Date fourthDate(3,7,2020);
  Date fifthDate(3,1,2020);
  Date sixthDate(4,1,2020);
  cout << "Initialization" << endl;
  Date seventhDate = sixthDate; // initialization (assignment operator is
not invoked
  Date eigthDate(seventhDate); //initialization (assignment operator is not
invoked
  cout << "Assignment" << endl;
  eigthDate=firstDate;
```

```
// test overload <
  cout << "\ntest of overloaded < operator" << endl;

  if (firstDate < secondDate)
      {
            firstDate.display();
            cout << " is less than ";
            secondDate.display();
            cout << endl;
      }
  else
      {
            secondDate.display();
            cout << " is less than ";
            firstDate.display();
            cout << endl;
      }

      if (firstDate < thirdDate)
      {
      firstDate.display();
            cout << " is less than ";
            thirdDate.display();
            cout << endl;
      }
  else
      {
            thirdDate.display();
            cout << " is less than ";
            firstDate.display();
            cout << endl;
      }

       if (firstDate < fourthDate)
      {
            firstDate.display();
            cout << " is less than ";
            fourthDate.display();
            cout << endl;
      }
  else
      {
            fourthDate.display();
            cout << " is less than ";
            firstDate.display();
            cout << endl;
      }

       if (firstDate < fifthDate)
```

```
        {
              firstDate.display();
              cout << " is less than ";
              fifthDate.display();
              cout << endl;
        }
    else
        {
              fifthDate.display();
              cout << " is less than ";
              firstDate.display();
              cout << endl;
        }

         if (firstDate < sixthDate)
        {
        firstDate.display();
              cout << " is less than ";
              sixthDate.display();
              cout << endl;
        }
    else
        {
              sixthDate.display();
              cout << " is less than ";
              firstDate.display();
              cout << endl;
        }

// test overloaded >
    cout << "\ntest of overloaded > operator" << endl;
        if (firstDate > secondDate)
        {
              firstDate.display();
              cout << " is greater than ";
              secondDate.display();
              cout << endl;
        }
    else
        {
              secondDate.display();
              cout << " is greater than ";
              firstDate.display();
              cout << endl;
        }

        if (firstDate > thirdDate)
        {
        firstDate.display();
              cout << " is greater than ";
```

```
            thirdDate.display();
            cout << endl;
        }
    else
        {
            thirdDate.display();
            cout << " is greater than ";
            firstDate.display();
            cout << endl;
        }


         if (firstDate > fourthDate)
        {
            firstDate.display();
            cout << " is greater than ";
            fourthDate.display();
            cout << endl;
        }
    else
        {
            fourthDate.display();
            cout << " is greater than ";
            firstDate.display();
            cout << endl;
        }

         if (firstDate > fifthDate)
        {
            firstDate.display();
            cout << " is greater than ";
            fifthDate.display();
            cout << endl;
        }
    else
        {
            fifthDate.display();
            cout << " is greater than ";
            firstDate.display();
            cout << endl;
        }

    if (firstDate > sixthDate)
        {
        firstDate.display();
            cout << " is greater than ";
            sixthDate.display();
            cout << endl;
        }
    else
```

```
    {
          sixthDate.display();
          cout << " is greater than ";
          firstDate.display();
          cout << endl;
    }

    // test overloaded ==
cout << "\ntest of overloaded == operator" << endl;
if (firstDate == secondDate)
    {
          firstDate.display();
          cout << " is equal to ";
          secondDate.display();
          cout << endl;
    }
else
    {
          secondDate.display();
          cout << " is not equal to ";
          firstDate.display();
          cout << endl;
    }

    thirdDate=firstDate;
    if (firstDate == thirdDate)
    {
    firstDate.display();
          cout << " is equal to ";
          thirdDate.display();
          cout << endl;
    }
else
    {
          thirdDate.display();
          cout << " is equal to ";
          firstDate.display();
          cout << endl;
     }

    // test overloaded +
    cout << "\ntest of overloaded + operator" << endl;
    Date birthday(11,24,2019);
    birthday.display();
    cout << endl;

    Date anotherDay;
    anotherDay.display();
    cout << endl;
```

```
    anotherDay = birthday + 5  ;
    anotherDay.display();
    cout << endl;

    anotherDay = anotherDay + 30  ;
    anotherDay.display();
    cout << endl;

    anotherDay = anotherDay + 30  ;
    anotherDay.display();
    cout << endl;
// anotherDay = 30 + anotherDay  ; compilation error

    // test overloaded +=
    cout << "\ntest of overloaded += operator" << endl;

    anotherDay.display();
    cout << endl;

    anotherDay += 20  ;
    anotherDay.display();
    cout << endl;

//  anotherDay += anotherDay + 20;  // not defined

    // test overloaded prefix ++
    cout << "\ntest of overloaded prefix ++ operator" << endl;

    anotherDay.display();
    cout << endl;

     ++anotherDay ;
    cout << anotherDay;

    // test overloaded postfix ++
    cout << "\ntest of overloaded postfix ++ operator" << endl;

     anotherDay++ ;
    cout << anotherDay;

    // test of assignment operator
    cout << "\ntest of overloaded = operator" << endl;
    seventhDate = sixthDate = fifthDate;
    seventhDate.display();
    cout << endl;
    sixthDate.display();
    cout << endl;
    fifthDate.display();
    cout << endl;
```

```
	// test overloaded >>
	cout << "\ntest of overloaded >> operator" << endl;
	cin >> birthday;

	// test overloaded <<
	cout << "\ntest of overloaded << operator" << endl;
	cout << birthday;
}
```

output

```
non-default constructor called
non-default constructor called
non-default constructor called
non-default constructor called
non-default constructor called
non-default constructor called
Initialization
copy constructor called
copy constructor called
Assignment
copy assign operator invoked

test of overloaded < operator
2/6/2019 is less than 3/6/2020
3/5/2020 is less than 3/6/2020
3/6/2020 is less than 3/7/2020
3/1/2020 is less than 3/6/2020
3/6/2020 is less than 4/1/2020

test of overloaded > operator
3/6/2020 is greater than 2/6/2019
3/6/2020 is greater than 3/5/2020
3/7/2020 is greater than 3/6/2020
3/6/2020 is greater than 3/1/2020
4/1/2020 is greater than 3/6/2020

test of overloaded == operator
2/6/2019 is not equal to 3/6/2020
copy assign operator invoked
3/6/2020 is equal to 3/6/2020

test of overloaded + operator
non-default constructor called
11/24/2019
```

```
default constructor called
0/0/0
copy constructor called
copy assign operator invoked
11/29/2019
copy constructor called
copy assign operator invoked
12/29/2019
copy constructor called
copy assign operator invoked
1/28/2020

test of overloaded += operator
1/28/2020
copy constructor called
copy assign operator invoked
copy constructor called
2/17/2020

test of overloaded prefix ++ operator
2/17/2020
copy constructor called
copy assign operator invoked
copy constructor called
2/18/2020

test of overloaded postfix ++ operator
copy constructor called
copy constructor called
copy assign operator invoked
2/19/2020

test of overloaded = operator
copy assign operator invoked
copy assign operator invoked
3/1/2020
3/1/2020
3/1/2020

test of overloaded >> operator

Enter month: 12

Enter day: 12

Enter year: 2012
```

```
test of overloaded << operator
12/12/2012
```

```
// overload3.cpp
// overloaded '+' operator adds two Distances
#include <iostream>
using namespace std;

class Distance                  // English Distance class
  {
  private:
    int feet;
    float inches;
  public:
       Distance()                // constructor (no args)
        { feet = 0; inches = 0.0; }
    Distance(int ft, float in)  // constructor (two args)
        { feet = ft; inches = in; }

    void getdist()              // get length from user
        {
        cout << "\nEnter feet: ";  cin >> feet;
        cout << "Enter inches: ";  cin >> inches;
        }

    void showdist()             // display distance
        { cout << feet << "\'-" << inches << '\"'; }

    Distance operator + ( Distance );  // add two distances
  };
                                // add this distance to d2
Distance Distance::operator + (Distance d2)   // return the sum
  {
  int f = feet + d2.feet;       // add the feet
  float i = inches + d2.inches;  // add the inches
  if(i >= 12.0)                 // if total exceeds 12.0,
    {                           // then decrease inches
    i -= 12.0;                  // by 12.0 and
```

```
    f++;                    // increase feet by 1
    }                      // return a temporary Distance
  return Distance(f,i);          // initialized to sum
  }

int main()
  {
  Distance dist1, dist3, dist4;   // define distances
  dist1.getdist();               // get dist1 from user

  Distance dist2(11, 6.25);       // define, initialize dist2

  dist3 = dist1 + dist2;          // single '+' operator

  dist4 = dist1 + dist2 + dist3;  // multiple '+' operators

                                  // display all lengths
  cout << "\ndist1 = ";  dist1.showdist();
  cout << "\ndist2 = ";  dist2.showdist();
  cout << "\ndist3 = ";  dist3.showdist();
  cout << "\ndist4 = ";  dist4.showdist();
       }
output
Enter feet: 10
Enter inches: 4

dist1 = 10'-4"
dist2 = 11'-6.25"
dist3 = 21'-10.25"
dist4 = 43'-8.5"
```

//Overload4

```
#include<iostream>
#include<stdio.h>

using namespace std;

class Test
{
    public:
    Test() {}
```

```
    Test(const Test &t)
    {
        cout<<"Copy constructor called "<<endl;
    }

    Test& operator = (const Test &t)
    {
        cout<<"Assignment operator called "<<endl;
        return *this;
    }
};

// Driver code
int main()
{
    Test t1, t2;
    t2 = t1;
    Test t3 = t1;
    getchar();
    return 0;
}
```

Output

```
Assignment operator called
Copy constructor called
```