```
Script started on Sun 13 May 2012 10:52:38 AM CDT
\033]0;georgia@georgia-MT6017: ~/cplusplus\007georgia@georgia-MT6017:~/cplusplus$ C
PP --version
This is CPP version 1.219 executing under perl v5.12.4 and compiling with:

g++ (Ubuntu/Linaro 4.6.1-9ubuntu3) 4.6.1
Copyright (C) 2011 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.


\033]0;georgia@georgia-MT6017: ~/cplusplus\007georgia@georgia-MT6017:~/cplusplus$ p
wd
/home/georgia/cplusplus
\033]0;georgia@georgia-MT6017: ~/cplusplus\007georgia@georgia-MT6017:~/cplusplus$ c
at complex.info
Name: Jakob Hansen
Class: CSC121 - Evening Section
Lab: Complex Number ADT

Levels attempted:

5 - Defined a class to represent complex numbers, put it in a library, included
methods for all basic arithmetic plus conjugate and magnitude, as well as i/o
methods, constructors, accessors, mutators. For what its worth I added a method
for reading in entire expressions, like a+bi / c-di.

+2 - For inline'ing where possible, and constifying method definitions where
appropriate. Oh yeah and constructor initialization lists.



Program Description:

A program that allows the user to do basic math with complex numbers. A menu
present the user with the options of printing the results of the four arithmetic
operations on two complex numbers, calculating the results of single
expressions, or printing the magnitude and conjugate of a single complex number.
The user must enter the complex numbers in proper form (a+bi) and the results
are displayed as such.

\033]0;georgia@georgia-MT6017: ~/cplusplus\007georgia@georgia-MT6017:~/cplusplus$ c
at complex.h
#ifndef COMPLEX_H_INC
#define COMPLEX_H_INC

#include <iostream>
#include <cmath>
#include <cctype>

using namespace std;

/* Defines a type to represent complex numbers, expressed as the sum (or
difference) of two numbers: one real and one imaginary. The imaginary half is
denoted by the symbol 'i' which indicates multiplication by the impossible
square root of -1. There is a read method for individual complex numbers, as
well as one for whole expressions. The four main arithmetic operations, plus
magnitude and conjugate are defined. */

class complexNum
{
    double rl, imgnry;

public:
    // constructors
    complexNum(void) : rl(0.0), imgnry(0.0) { }
    complexNum(const complexNum & num) : rl(num.rl), imgnry(num.imgnry){ }
    complexNum(double new_rl, double new_imgnry) : rl(0.0), imgnry(0.0)
    {
        set_rl(new_rl);
        set_imgnry(new_imgnry);
    }

    // printing & reading
    void print(void) const;
    bool read(void);
    bool read_expr(void); // reads in an entire expression ie: 4+5i + 3+6i

    // addition
    complexNum plus(const complexNum & num) const
        {return complexNum(rl + num.rl, imgnry + num.imgnry);}

    // subtraction
    complexNum minus(const complexNum & num) const
        {return complexNum(rl - num.rl, imgnry - num.imgnry);}

    // multiplication
    complexNum mult_by(const complexNum & num) const
        {return complexNum(rl * num.rl - imgnry * num.imgnry,
                           rl * num.imgnry + imgnry * num.rl);}

    // division
    complexNum div_by(const complexNum & num) const
        {return complexNum((rl * num.rl + imgnry * num.imgnry) /
                    (num.rl * num.rl + num.imgnry * num.imgnry),
                    (rl * num.imgnry - imgnry * num.rl) /
                    (num.rl * num.rl + num.imgnry * num.imgnry));}

    // magnitude  (returns double)
    double mag(void) const {return sqrt(rl * rl + imgnry * imgnry);}

    // conjugate
    complexNum conj(void) const {return complexNum(rl, -imgnry);}

    // accessors
    double get_rl(void) const {return rl;}
    double get_imgnry(void) const {return imgnry;}

    // mutators
    void set_rl(double new_rl)
    {
        rl = new_rl;
        return;
    }
    void set_imgnry(double new_imgnry)
    {
        imgnry = new_imgnry;
        return;
    }
};

#endif
\033]0;georgia@georgia-MT6017: ~/cplusplus\007georgia@georgia-MT6017:~/cplusplus$ c
at complex.cpp
#include <iostream>
#include <cmath>
#include <climits>
#include <cctype>
#include "complex.h"

using namespace std;

void complexNum::print(void) const // prints numbers with appropriate notation
{
    if (imgnry > 0)
    {
        cout << rl << '+' << imgnry << 'i';
    }
```

```
    else if (fabs(imgnry) <= pow(10, -6))
    {
        cout << rl;
    }
    else
    {
        cout << rl << imgnry << 'i';
    }
    return;
}

/* Reads complex numbers, with a check for correct operator between real and
imaginary parts. Numeric input failure is reported to the caller, but dealing
with it and re-prompting is up to the caller. */

bool complexNum::read(void)
{
    char sign, t;
    bool success;

    cin >> rl >> sign >> imgnry >> t;

    success = !cin.fail();

    if (sign == '-')
    {
        imgnry = -imgnry;
    }
    else if (sign != '-' && sign != '+')
    {
        success = false;
        cout << "\nUnrecognized operator!!";
    }

    return success;
}

/* Reads in an entire expression, */

bool complexNum::read_expr(void)
{
    bool success;
    complexNum num1;
    complexNum num2;
    complexNum result;
    char oprtr;

    num1.read();
    cin >> oprtr;
    num2.read();

    success = !cin.fail();

    switch (oprtr)
    {
        case '+':
        result = num1.plus(num2);
        break;
        case '-':
        result = num1.minus(num2);
        break;
        case '*':
        result = num1.mult_by(num2);
        break;
        case '/':
        result = num1.div_by(num2);
        break;
        default:
        cout << "\nInvalid Operator!"
             << "\nSupported operators are '+', '-', '*', and '/'.";
        success = false;
    }

    rl = result.get_rl();
    imgnry = result.get_imgnry();

    return success;
}
\033]0;georgia@georgia-MT6017: ~/cplusplus\007georgia@georgia-MT6017:~/cplusplus$ c
at complexnumbers.cpp
#include <iostream>
#include <cmath>
#include <cctype>
#include <climits>
#include "complex.h"

using namespace std;

int main(void)
{
    complexNum firstnumber, secondnumber;
    char yesno, choice;

    cout << "\n\tWelcome to the Complex Number Calculator!!!";

    do
    {
        yesno = 'n';
        cout << "\nHow would you like to proceed?"
             << "\n1. (P)rint sum, difference, product, and quotient of two"
             << " complex numbers."
             << "\n2. (E)nter in your own expression."
             << "\n3. (F)ind the magnitude and conjugate of a complex number."
             << "\n4. (Q)uit\n";

        cin >> choice;
        cin.ignore(INT_MAX, '\n');

        if (toupper(choice) == 'P' || choice == '1')
        {
            cout << "\nPlease enter the first complex number:  ";
            while( !firstnumber.read())
            {
                cin.clear();
                cin.ignore(INT_MAX, '\n');
                cout << "\nInvalid notation! Please try again!\n";
                cout << "\nPlease enter the first complex number:  ";
            }

            cout << "\nPlease enter the second complex number:  ";
            while( !secondnumber.read())
            {
                cin.clear();
                cin.ignore(INT_MAX, '\n');
                cout << "\nInvalid notation! Please try again!\n";
                cout << "\nPlease enter the second complex number:  ";
            }

            cout << "\nYou entered: ";
            firstnumber.print();
            cout << " and ";
            secondnumber.print();

            cout << "\n\nSum: ";
            firstnumber.plus(secondnumber).print();

            cout << "\nDifference: ";
            firstnumber.minus(secondnumber).print();
```

```
            cout << "\nProduct: ";
            firstnumber.mult_by(secondnumber).print();

            cout << "\nQuotient: ";
            firstnumber.div_by(secondnumber).print();

            cout << "\nWould you like to continue?  ";
            cin >> yesno;
        }
        if (toupper(choice) == 'E' || choice == '2')
        {
            cout << "\nSupported operators are '+', '-', '*', and '/'."
                 << "\nPlease enter an expression in the form a+bi + c+di:";

            while (!firstnumber.read_expr())
            {
                cin.clear();
                cin.ignore(INT_MAX, '\n');
                cout << "\nInvalid notation! Please try again."
                     << "\nEnter an expression in the form a+bi + c+di: ";
            }

            cout << "\nThe answer is: ";
            firstnumber.print();

            cout << "\nWould you like to continue?  ";
            cin >> yesno;
            cin.ignore(INT_MAX, '\n');
        }
        if (toupper(choice) == 'F' || choice == '3')
        {
            cout << "\nPlease enter a complex number:  ";
            firstnumber.read();

            cout << "\nThe magnitude of ";
            firstnumber.print();
            cout << " is " << firstnumber.mag();

            cout << "\nThe conjugate is ";
            firstnumber.conj().print();

            cout << "\nWould you like to continue?  ";
            cin >> yesno;
            cin.ignore(INT_MAX, '\n');
        }
    } while (toupper(yesno) == 'Y');

    cout << "\nGoodbye!\n\n";

    return 0;
}
\033]0;georgia@georgia-MT6017: ~/cplusplus\007georgia@georgia-MT6017:~/cplusplus$ C
PP complex complexnumbers
complex.cpp...
complexnumbers.cpp***


\033]0;georgia@georgia-MT6017: ~/cplusplus\007georgia@georgia-MT6017:~/cplusplus$ .
/complexnumbers.out

       Welcome to the Complex Number Calculator!!!
How would you like to proceed?
1. (P)rint sum, difference, product, and quotient of two complex numbers.
2. (E)nter in your own expression.
3. (F)ind the magnitude and conjugate of a complex number.
4. (Q)uit
p

Please enter the first complex number:  4+3i

Please enter the second complex number:  wmnevhi

Invalid notation! Please try again!

Please enter the second complex number:  4-2i

You entered: 4+3i and 4-2i

Sum: 8+1i
Difference: 0+5i
Product: 22+4i
Quotient: 0.5-1i
Would you like to continue?  y

How would you like to proceed?
1. (P)rint sum, difference, product, and quotient of two complex numbers.
2. (E)nter in your own expression.
3. (F)ind the magnitude and conjugate of a complex number.
4. (Q)uit
p

Please enter the first complex number:  4+4i

Please enter the second complex number:  4+4i

You entered: 4+4i and 4+4i

Sum: 8+8i
Difference: 0
Product: 0+32i
Quotient: 1
Would you like to continue?  y

How would you like to proceed?
1. (P)rint sum, difference, product, and quotient of two complex numbers.
2. (E)nter in your own expression.
3. (F)ind the magnitude and conjugate of a complex number.
4. (Q)uit
e

Supported operators are '+', '-', '*', and '/'.
Please enter an expression in the form a+bi + c+di:4+3i / 6-7i

The answer is: 0.0352941-0.541176i
Would you like to continue?  y

How would you like to proceed?
1. (P)rint sum, difference, product, and quotient of two complex numbers.
2. (E)nter in your own expression.
3. (F)ind the magnitude and conjugate of a complex number.
4. (Q)uit
e

Supported operators are '+', '-', '*', and '/'.
Please enter an expression in the form a+bi + c+di:4+3i gsgf 3-6i

Invalid Operator!
Supported operators are '+', '-', '*', and '/'.
Invalid notation! Please try again.
Enter an expression in the form a+bi + c+di: 3-5i * 8 -4i

The answer is: 4-52i
Would you like to continue?  y

How would you like to proceed?
1. (P)rint sum, difference, product, and quotient of two complex numbers.
2. (E)nter in your own expression.
```

```
3. (F)ind the magnitude and conjugate of a complex number.
4. (Q)uit
f

Please enter a complex number:  34-8i

The magnitude of 34-8i is 34.9285
The conjugate is 34+8i
Would you like to continue?  n

Goodbye!

\033]0;georgia@georgia-MT6017: ~/cplusplus\007georgia@georgia-MT6017:~/cplusplus$ c
at complex.tpq
1. Why do your class methods take fewer arguments than you might/would expect?
    Well I initially wrote the methods with twice as many arguments as needed,
    forgetting that the member variables without a dot (.) operator refer to the
    CALLING object.

2. Does your addition method change the value of the calling object? Should it?
Does the compiler change x when you have 'x + y' in your program? What about y?
Does/Should your addition method change the other complex number?
Does this extend to the other operations?
    No, none of the arithmetic methods change the value of the calling object.
    The compiler doesn't change the value of any operands with the built in
    types when doing arithmetic, as this would make programming ridiculously
    useless! Arithmetic methods should return a new value without changing
    either the calling object or the object passed to the method.

3. What kind of value should be returned from the standard math operations (i.e.
what TYPE of value)? From conjugate? From magnitude?
    All of the standard math operations return complex number types, except
    magnitude, which returns a double.

4. Does your input method prompt the user? Why shouldn't it?
    No, it doesn't prompt, and it shouldn't because the more junk your methods
    output to the screen, the less re-usable they become. And other people don't
    want to read your junk messages they just want the thing to work!

5. Does your output method print anything besides the complex number (using
proper notation) -- even an endl? Why shouldn't it?
    Nope. Just prints the number. See answer to #4.

More tpqs:

1. Should any of your methods be inline'd for speed? Which ones will you inline?
How does one do this for a class method?
    All of my methods are inlined, except for print() and read(). The math for
    all of the other methods is real simple so they are all inlined. To inline
    class methods you just define them where you otherwise would prototype them,
    no need to put inline beforehand.

2. If you end up inline'ing all of your methods, do you still need an
implementation file for your library? Why/Why not?
    I suppose you would, actually, if you had anything else besides methods to
    define...and I don't know how you would compile it, since the compiler
    only takes .cpp files...

3. Do the accessors, output, or math operation methods alter the calling object
in any way? How do you specify such a situation to the compiler so it can better
handle/enforce your decision?
    No they do not, see question 2 in the first round of questions. To specify
    this, you place the const keyword after the head of the function but before
    the code that actually defines it. This way the compiler is guaranteed not
    to change the value of the calling object.

4. What needs to be in your constructor initializer lists? Is there a certain
order to the list? Does it go on the prototype or the definition?
    The values in the constructor initialization list are whatever you want the
    objects to be initialized as, seems most likely that these would usually
    be the origin values for that particular datatype. I don't think there is
    a particular order that needs to be adhered to? They go in the prototype,
    after which the definition is just a pair of empty braces except for the
    parameterized initialization.
\033]0;georgia@georgia-MT6017: ~/cplusplus\007georgia@georgia-MT6017:~/cplusplus$ e
xit
exit

Script done on Sun 13 May 2012 10:56:03 AM CDT
```