```
/*
 * advanced programming
 * Alireza Akhavan Pour
 * akhavan@alirezaWeb.com
 * date: 1394/12/17
 */
int main()
{
    cout<< "In the name of God";
    Lecture_12();
    return 0;
}

string Lecture_12()
{
    cout << "1.const" <<endl;
    cout << "2.overload" <<endl;
    return ":)"
}
```

Shahid Rajaee Teacher Training University

1

اعضای ثابت کلاس – Const member functions

```
// constructor on const object
#include <iostream>
using namespace std;

class MyClass {
  public:
    int x;
    MyClass(int val) : x(val) {}
    int get() {return x;}
};

int main() {
  const MyClass foo(10);
// foo.x = 20;            // not valid: x cannot be modified
  cout << foo.x << '\n';  // ok: data member x can be read
  return 0;
}
```

در اینجا به هیچ کدام از متدها نمی‌توان دسترسی داشت.
حتی متد get، چون موقع ساخت شئ آن را const کردیم.

اما واقعا به متد get نیاز داریم...
✓ راه حل:

```
int get() const {return x;}
```

Try Online | 1-const-member-function.cpp

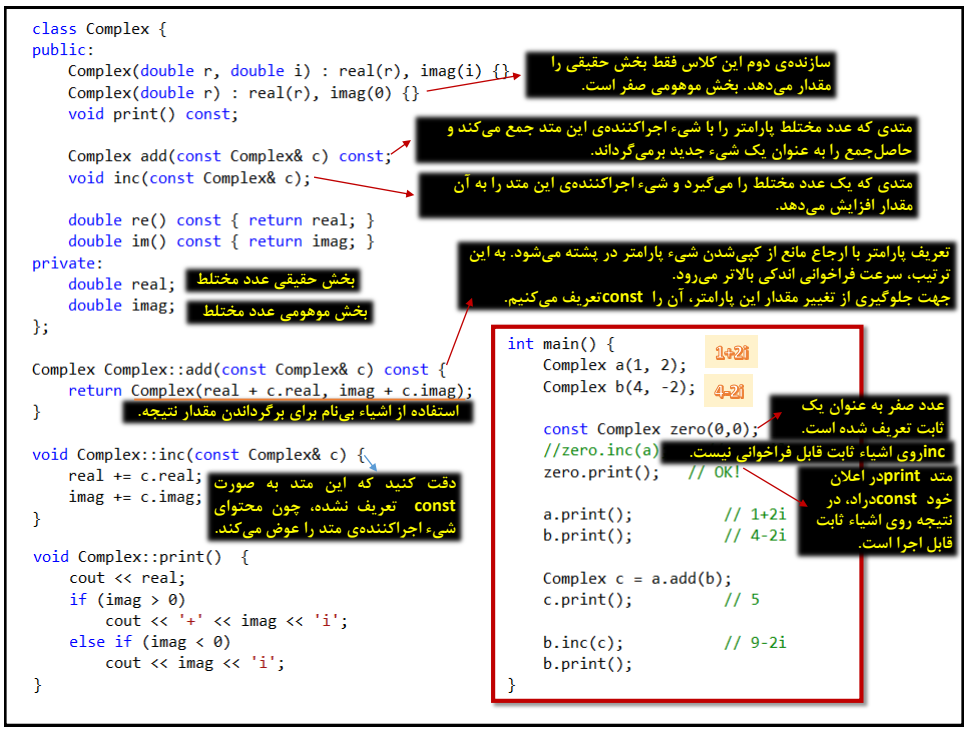
2

اعداد مختلط بدون سربارگذاری عملگرها

در این بخش کلاس Complex برای استفاده از اعداد مختلط تعریف می‌شود. اعمال جمع و افزایش از طریق متدهای عادی قابل اعمال روی اعداد هستند.

3_ComplexNonOverload.cpp

3



سربارگذاری عملگرهای + و +=

در این بخش، دو عملگر + و += برای کلاس اعداد مختلط سربارگذاری می‌شوند. با داشتن این عملگرها، این کلاس به شکل خواناتر و راحت‌تری قابل استفاده است.

4_ComplexAddInc.cpp

5

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

class Complex {
public:
    Complex(double r, double i) : real(r), imag(i) {}
    Complex(double r) : real(r), imag(0) {}

    void print() const;

//  Complex add(const Complex& c) const;
    Complex operator+(const Complex& c) const;
//  void inc(const Complex& c);
    Complex& operator+=(const Complex& c);

    double re() const { return real; }
    double im() const { return imag; }

private:
    double real;
    double imag;

};
```

تعریف علگر +. دقت کنید که این تعریف دقیقاً مانند تعریف متد **add**است، فقط نام **add**جای خود را به **operator** + داده است.

این عملگر بسیار شبیه متد **inc** تعریف شده با یان فرق که به عنوان نتیجه شیء سمت چپ عملگر را برمی‌گرداند تا با تعریف عملگر += در زبان **C**مشابه عمل کند.

```
//Complex Complex::add(const Complex& c) const
Complex Complex::operator+(const Complex& c) const
{
    return Complex(real + c.real, imag + c.imag);
}

//void Complex::inc(const Complex& c)
Complex& Complex::operator+=(const Complex& c)
{
    real += c.real;
    imag += c.imag;
    return *this;
}

void Complex::print() const
{
    cout << real;
    if (imag > 0)
        cout << '+' << imag << 'i';
    else if (imag < 0)
        cout << imag << 'i';
}
```

برگرداندن شیء اجراکننده‌ی متد (در اینجا، شیء سمت چپ عملگر)

```
int main()
{
    Complex a(1, 2);
    Complex b(4, -2);

    // Complex c = a.add(b);
    Complex c = a + b;
    c.print();          // 5

    // b.inc(c);
    b += c;
    b.print();          // 9-2i
    b.operator+(c);
    b+=c;
    (b += c).print();
}
```

استفاده از عملگر + به جای متد add

برای فراخوانی عملگرها استفاده از این شکل هم ممکن است. دقت کنید که این نحوه‌ی استفاده با شکل تعریف عملگر نیز همسان است.

این که عملگر += مقدار شیء سمت چپ را برمی‌گرداند اجازه می‌دهد جملاتی شبیه این داشته باشیم.

بعد از اجرای

c = (b += a) += a;

مقدار a، b و c چه خواهد بود؟

8

سربارگذاری عملگرهای + برای جمع با یک عدد حقیقی

در این بخش، عملگر + برای جمع یک عدد مختلط با یک عدد حقیقی سربارگذاری شده است. به این ترتیب، می‌توان عباراتی مانند c + 2.5 داشت که در آن c یک عدد مختلط است.

5_ComplexAddDouble.cpp


```
Complex Complex::operator+(double r) const
{
    return Complex(real + r, imag);
}
```

```
int main()
{
    Complex a(1, 2);

    Complex c = a + 2;
    c.print();              // 3+2i
}
```

آیا میتوانیم به جای این خط بنویسیم

c=2+a;

چرا؟

راه کارتان برای مجاز شدن آن چیست؟

عملگر + در حالی که سمت چپ آن عدد حقیقی است

در این بخش، عملگر + برای جمع یک عدد مختلط با یک عدد حقیقی سربارگذاری شده است. به این ترتیب، می‌توان عباراتی مانند ۲.۵ + c داشت که در آن cیک عدد مختلط است.

6_ComplexAddDouble.cpp

11

دقت کنید که این عملگر به شکل یک تابع معمولی تعریف شده نه به عنوان عضوی از کلاس Complex

```
Complex operator+(const double r, const Complex& c)
{
    return Complex(c.re() + r, c.im());
}
```

چون این تابع خارج از کلاس Complexتعریف شده، دسترسی به فیلدهای شیء c باید از طریق متدهای عمومی صورت پذیرد.

```
int main()
{
    Complex a(1, 2);
    Complex c = 2 + a;
    c.print();              // 3+2i
}
```

معادل با
Complex c = operator+(2, a)

12

عملگر >> برای درج در خروجی

در این بخش، عملگر >> برای درج یک عدد مختلط در یک جریان خروجی سربارگذاری می‌شود.

7_ComplexInsertion.cpp

13

شکل استاندارد تعریف اپراتور درج در خروجی.
کلاس ostream نماینده‌ی یک جریان خروجی است.

```
ostream& operator<<(ostream& out, const Complex& c)
{
    out << c.re();
    if (c.im() > 0)
        out << '+' << c.im() << 'i';
    else if (c.im() < 0)
        out << c.im() << 'i';

    return out;
}
```

چون این تابع خارج از کلاس Complexتعریف شده، دسترسی به فیلدهای شیء cباید از طریق متدهای عمومی صورت پذیرد.

نتیجه‌ی این تابع، جریان خروجی سمت چپ عملگر است تا بتوان عباراتی مانند cout << a << b را نوشت.

```
int main()
{
    Complex a(1, 2);
    Complex b(4, -2);

    cout << a << b << endl;

    system ("pause");
}
```

coutیک شیء سراسری از نوع ostreamاست که با includeکردن iostreamدر اختیار ما قرار می‌گیرد.

اگر بخواهیم این خط را به شیوه‌ی فراخوانی operator<<(...) بنویسیم چگونه خواهد شد؟

14

توابع دوست

در این بخش، عملگر >> به عنوان دوست کلاس Complexتعریف می‌شود و در نتیجه می‌تواند به اعضای خصوصی این کلاس دسترسی داشته باشد. طبیعتاً چنین کاری برخلاف اصول برنامه‌نویسی شیءگرا است و مخفی‌سازی اطلاعات را زیر سؤال می‌برد. به همین دلیل استفاده از این امکان جز در موارد خاص توصیه نمی‌شود. یکی از این موراد سربارگذاری عملگر >> است.

8_ComplexFriend.cpp

15

```
class Complex {
public:
    Complex(double r, double i) : real(r), imag(i) {}
    Complex(double r) : real(r), imag(0) {}

    void print() const;

    Complex operator+(const Complex& c) const;
    Complex& operator+=(const Complex& c);
    Complex operator+(double r) const;

    double re() const { return real; }
    double im() const { return imag; }

    friend ostream& operator<<(ostream& out, const Complex& c);
private:
    double real;
    double imag;
};
```

تابع operator<<می‌تواند به اعضای خصوصی این کلاس دسترسی داشته باشد.

```
ostream& operator<<(ostream& out, const Complex& c)
{
    out << c.real;
    if (c.imag > 0)
        out << '+' << c.imag << 'i';
    else if (c.imag < 0)
        out << c.imag << 'i';

    return out;
}
```

چون این تابع «دوست» کلاس Complexتعریف شده، می‌تواند به اعضای خصوصی آن دسترسی داشته باشد.

16

منابع این جلسه

- ✓http://www.cplusplus.com/doc/tutorial/templates/
- ✓http://ramtung.ir/apnotes/html/09_OperOver.html

17