BCC702 - Programação de Computadores II

Containeres Associativos e Conjuntos Programação Orientada a Objetos

Profª Valéria de Carvalho Santos

Slides adaptados do profº Alan de Freitas

Relembrando... Templates

```
template <class T>
T maximo(T a, T b, T c){
    T max = a;
    if(b > max){
        max = b;
    }
    if(c > max){
        max = c;
    }
    return max;
}
```

Relembrando... Templates

```
template <class T>
T maximo(T a, T b, T c){
    T max = a;
    if(b > max){
        max = b;
    }
    if(c > max){
        max = c;
    }
    return max;
}
```

```
int main(){
    int a = 7, b = 9, c = 2;
    cout<<"int:"<<maximo(a, b, c)<< endl;

    cout<<"double:"<<maximo(7.2,3.8,6.5)<<endl;

    cout<<"char:"<<maximo('e', 'r', 'v')<<endl;
}
```

Relembrando... Templates

```
template <class T>
T maximo(T a, T b, T c){
    T max = a;
    if(b > max){
        max = b;
    }
    if(c > max){
        max = c;
    }
    return max;
}
```

```
int main(){
    int a = 7, b = 9, c = 2;
    cout<<"int:"<<maximo(a, b, c)<< endl;

    cout<<"double:"<<maximo(7.2,3.8,6.5)<<endl;

    cout<<"char:"<<maximo('e', 'r', 'v')<<endl;
}
```

```
int: 9
double: 7.2
char: v
```

Relembrando... STL

Containers	Gerenciam coleções de objetos Tipos: sequenciais e associativos
Iteradores	Percorrem elementos dos containers
Algoritmos	Processam elementos dos containers

Relembrando... Containers Sequenciais

- Vector
- Deque
- List

Relembrando... Vector

- Internamente, utiliza um vetor alocado dinamicamente
- Funções básicas:
  - *push_back*: insere no fim
  - *pop_back*: remove do fim
  - *front*: retorna o elemento do início
  - *back*: retorna o elemento do fim

Relembrando... Deque

- Internamente, utiliza um vetor alocado dinamicamente, com variáveis para controlar o início e o fim da sequência para implementar uma estrutura circular

Relembrando... Deque

- Funções básicas:
  - *push_front*: insere no início
  - *push_back*: insere no fim
  - *pop_front*: remove do início
  - *pop_back*: remove do fim
  - *front*: retorna o elemento do início
  - *back*: retorna o elemento do fim

Relembrando... List

- Representa listas duplamente encadeadas
- Internamente, utilizam células para armazenar valores com ponteiros para a célula anterior e a próxima célula

Relembrando... List

- Utiliza apenas a quantidade de memória necessária
- Funções básicas:
  - *push_front*: insere no início
  - *push_back*: insere no fim
  - *pop_front*: remove do início
  - *pop_back*: remove do fim
  - *front*: retorna o elemento do início
  - *back*: retorna o elemento do fim

Relembrando... Subscritos

- Permite o acesso aleatórios aos elementos do container
- Apenas para *vector* e *deque*

Relembrando... Iteradores

- Utilizados para percorrer qualquer container

```
#include <iostream>
#include <list>

using namespace std;

int main(){
    list<int> c;
    for(int i = 0; i < 10; i++)
        c.push_back(i+1);
    list<int>::iterator p;
    for(p = c.begin(); p != c.end(); p++)
        cout << *p << " ";
}
```

Containers Associativos e Conjuntos

- Os containers associativos são utilizados para representar conjuntos
- Foca em saber se um elemento pertence ou não ao conjunto
- A posição do elemento no conjunto não é relevante
- Como localizar um elemento no conjunto?
  - Chave de busca

Containers Associativos e Conjuntos

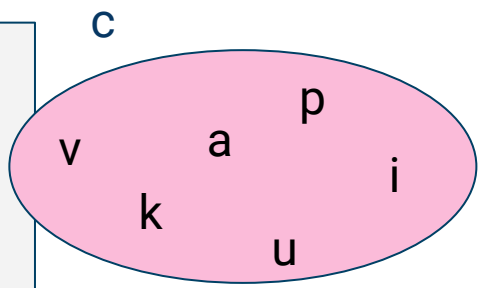
- Set
- Multiset
- Map

Set

- Permite o armazenamento de valores únicos
- Ao acessar cada elemento do *set*, eles são ordenados de acordo com seu valor

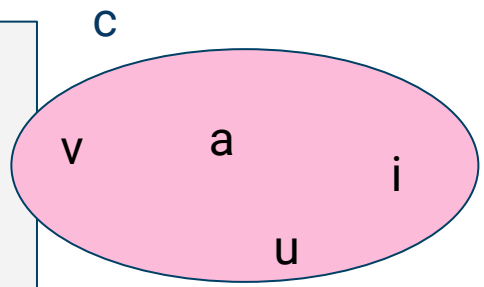
```
#include <iostream>
#include <set>
using namespace std;

int main(){
    set<char> c;

    c.insert('v');
    c.insert('a');
    c.insert('k');
    c.insert('p');
    c.insert('i');
    c.insert('u');

    set<char>::iterator p;
    for(p = c.begin(); p != c.end(); p++)
        cout << *p << " ";
}
```

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

int main(){
    set<char> c;

    c.insert('v');
    c.insert('a');
    c.insert('k');
    c.insert('p');
    c.insert('i');
    c.insert('u');

    set<char>::iterator p;
    for(p = c.begin(); p != c.end(); p++)
        cout << *p << " ";
}
```




```
a i k p u v
```

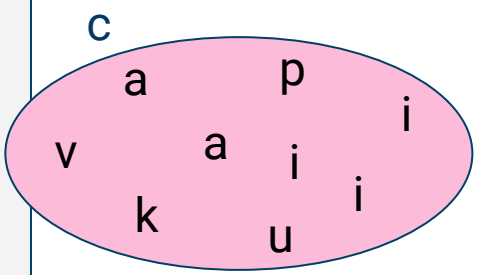
```
#include <iostream>
#include <set>
using namespace std;

int main(){
    set<char> c;

    c.insert('v');
    c.insert('a');
    c.insert('k');
    c.insert('p');
    c.insert('i');
    c.insert('u');

    c.erase('p');
    c.erase('k');
}
```



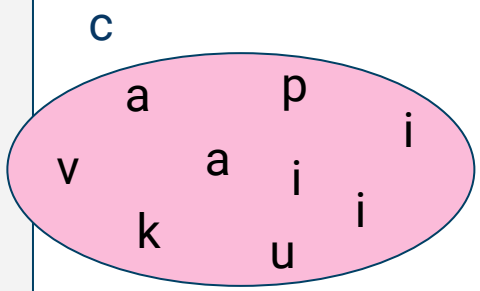
Multiset

- Similar ao *set*, sendo que permite armazenar elementos repetidos

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

int main(){
    multiset<char> c;

    c.insert('v');
    c.insert('a');
    c.insert('a');
    c.insert('k');
    c.insert('p');
    c.insert('i');
    c.insert('u');
    c.insert('i');
    c.insert('i');

    multiset<char>::iterator p;
    for(p = c.begin(); p != c.end(); p++)
        cout << *p << " ";
}
```



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

int main(){
    multiset<char> c;

    c.insert('v');
    c.insert('a');
    c.insert('a');
    c.insert('k');
    c.insert('p');
    c.insert('i');
    c.insert('u');
    c.insert('i');
    c.insert('i');

    multiset<char>::iterator p;
    for(p = c.begin(); p != c.end(); p++)
        cout << *p << " ";
}
```



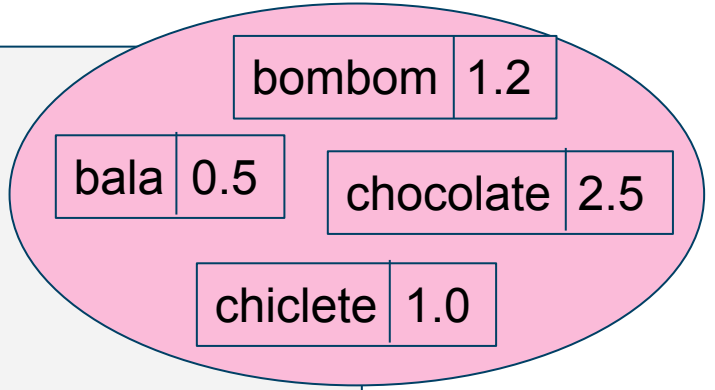

```
a a i i i k p u v
```

Map

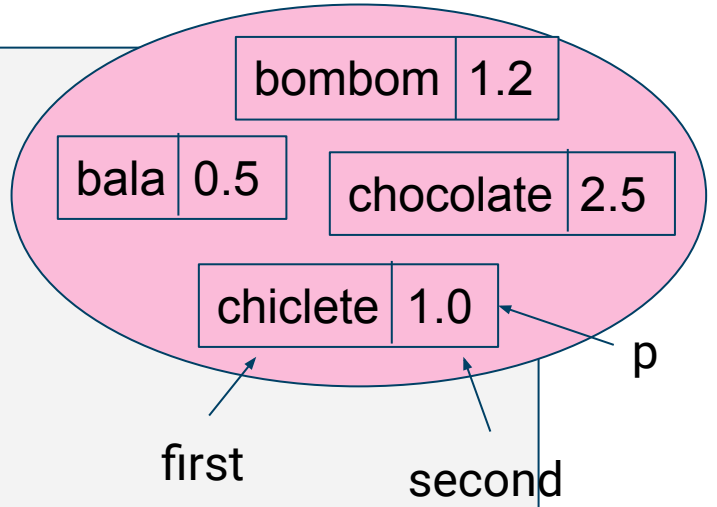
- Container que representa arranjos associativos
  - É composto por várias chaves, onde cada chave é associada a um valor
  - A chave e o valor não precisam ser do mesmo tipo

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

int main(){
    map<string,double> c;

    c["bala"] = 0.5;
    c["chocolate"] = 2.5;
    c["bombom"] = 1.2;
    c["chiclete"] = 1.0;


}
```

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

int main(){
    map<string,double> c;

    c["bala"] = 0.5;
    c["chocolate"] = 2.5;
    c["bombom"] = 1.2;
    c["chiclete"] = 1.0;


}
```



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

int main(){
    map<string,double> c;

    c["bala"] = 0.5;
    c["chocolate"] = 2.5;
    c["bombom"] = 1.2;
    c["chiclete"] = 1.0;

    map<string,double>::iterator p;
    for(p = c.begin(); p != c.end(); p++)
        cout << p->first << ": " << p->second << endl;
        //cout << (*p)->first << ": " << *(p)->second << endl;
}
```



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

int main(){
    map<string,double> c;

    c["bala"] = 0.5;
    c["chocolate"] = 2.5;
    c["bombom"] = 1.2;
    c["chiclete"] = 1.0;

    map<string,double>::iterator p;
    for(p = c.begin(); p != c.end(); p++)
        cout << p->first << ": " << p->second << endl;
        //cout << (*p).first << ": " << (*p).second << endl;
}
```

```
bala: 0.5
bombom: 1.2
chiclete: 1.0
Chocolate: 2.5
```

Programação Orientada a Objetos

- Abstração de dados
  - Permite modelar objetos do mundo real
  - Definição de classes
- Encapsulamento
  - Torna os detalhes de implementação invisível ao usuário
  - Protege a alteração indevida dos valores dos atributos

Sobrecarga de Operadores

- Já vimos que um mesmo operador pode ser utilizado para tipos diferentes
  - Operador + pode ser usado com int, float, double...
- A sobrecarga de operadores permite **redefinir a ação** de alguns operadores básicos para a classe em definição

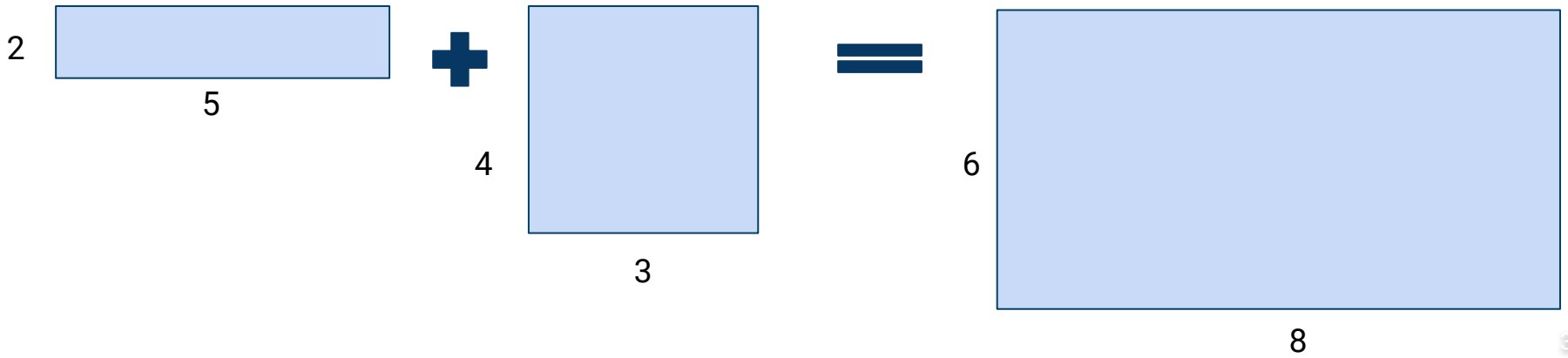
Sobrecarga de Operadores

- Precedência, associatividade e número de operandos não podem ser alterados pela sobrecarga.
- Alguns operadores que podem ser sobrecarregados:

```
-   *   /   =   <   >   == !=  <= >= ++ --  += -=  *= /= <<
>>      %   &&      ||
```

Sobrecarga de Operadores

- Exemplo: definir a sobrecarga do operador + para a classe Retângulo, sendo que a soma é a soma das larguras e das alturas dos dois retângulos somados.



```
class Retangulo{
    public:
        Retangulo(int, int);
        int area();
        Retangulo operator+(const Retangulo &);
    private:
        int largura, altura;
};

Retangulo Retangulo::operator+(const Retangulo &direita){
    int novaLargura = largura + direita.largura;
    int novaAltura = altura + direita.altura;
    Retangulo temp(novaLargura, novaAltura);
    return temp;
}

int main(){
    Retangulo r1(2,5);
    Retangulo r2(4,3);
    Retangulo r3 = r1 + r2;
    cout << r3.area();
}
```

```
class Retangulo{
    public:
        Retangulo(int, int);
        int area();
        Retangulo operator+(const Retangulo &);
    private:
        int largura, altura;
};

Retangulo Retangulo::operator+(const Retangulo &direita){
    int novaLargura = largura + direita.largura;
    int novaAltura = altura + direita.altura;
    Retangulo temp(novaLargura, novaAltura);
    return temp;
}

int main(){
    Retangulo r1(2,5);
    Retangulo r2(4,3);
    Retangulo r3 = r1 + r2;
    cout << "Area de r3: " << r3.area();
}
```

```
class Retangulo{
    public:
        Retangulo(int, int);
        int area();
        Retangulo operator+(const Retangulo &);
    private:
        int largura, altura;
};

Retangulo Retangulo::operator+(const Retangulo &direita){
    Retangulo temp(largura + direita.largura, altura + direita.altura);
    return temp;
}

int main(){
    Retangulo r1(2,5);
    Retangulo r2(4,3);
    Retangulo r3 = r1 + r2;
    cout << "Area de r3: " << r3.area();
}
```

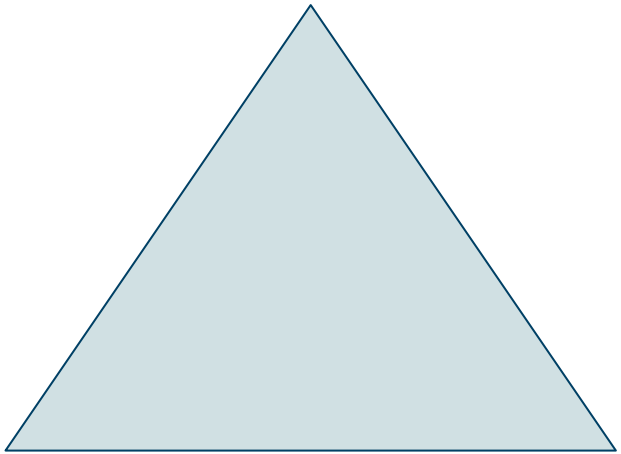
```
class Retangulo{
    public:
        Retangulo(int, int);
        int area();
        Retangulo operator+(const Retangulo &);
    private:
        int largura, altura;
};

Retangulo Retangulo::operator+(const Retangulo &direita){
    Retangulo temp(largura + direita.largura, altura + direita.altura);
    return temp;
}

int main(){
    Retangulo r1(2,5);
    Retangulo r2(4,3);
    Retangulo r3 = r1 + r2;
    cout << "Area de r3: " << r3.area();
}
```

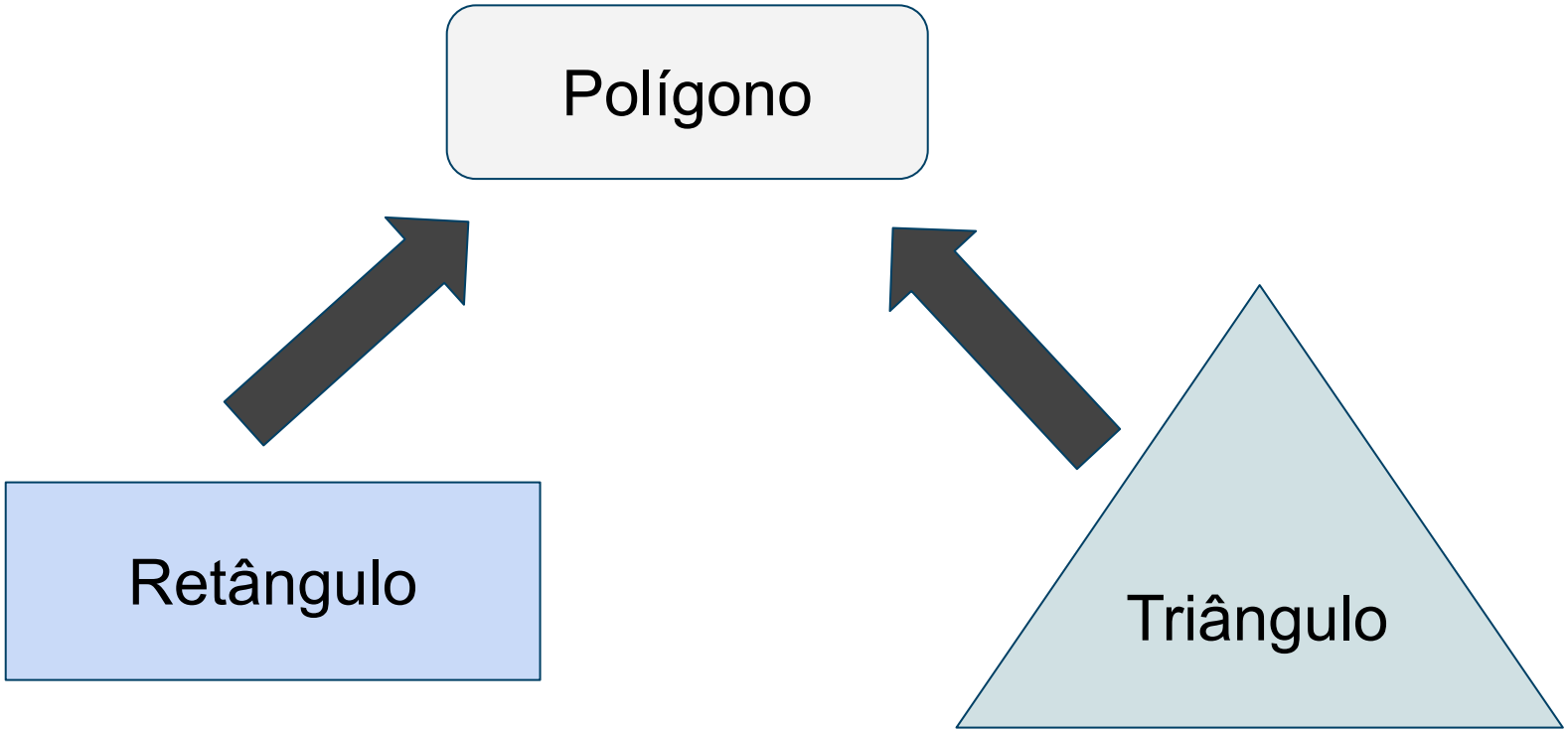
```
Area de r3: 48
```

Exercício

- Defina a sobrecarga do operador de igualdade (==) para retângulos, sendo que dois retângulos são iguais se possuem mesma largura e mesma altura.

- Observe as figuras abaixo.
  - Que atributos você definiria para cada uma? E que métodos?




Polígono
Retângulo
Triângulo

Herança

- Recurso de POO que permite o reúso de software através da criação de uma classe que absorve dados e comportamento de uma classe existente e os aprimora com novas capacidades.
- Ao criar uma classe, em vez de escrever atributos e funções-membro novos, é possível a nova classe **herdar** membros de uma classe existente.

Herança

- A classe existente é chamada **classe básica** ou **superclasse**
- A nova classe é chamada **classe derivada** ou **subclasse**
- Exemplos:

Superclasse	Subclasse
Polígono	Retângulo, Triângulo, Losango
Conta	Conta Corrente, Poupança, Conta Salário
Pessoa	Pessoa Física, Pessoa Jurídica

Herança

- Como implementar a herança em C++:
  - A superclasse é definida normalmente
  - A definição da subclasse deve informar de qual superclasse está herdando:

    class *nomeSubclasse* **:** public *nomeSuperclasse*

```
//representa poligonos que tem largura e altura
class Poligono{
    public:
        setValores(int, int);
    protected:
        int largura, altura;
};

class Retangulo : public Poligono{
    public:
        int area(){
            return largura*altura;
        }
};
```

```
//representa poligonos que tem largura e altura
class Poligono{
    public:
        setValores(int, int);
    protected:
        int largura, altura;
};

class Retangulo : public Poligono{
    public:
        int area(){
            return largura*altura;
        }
};
```

protected: os atributos não são acessíveis por quem cria objetos da classe; são acessíveis por objetos que herdam da classe.

```
//representa poligonos que tem largura e altura
class Poligono{
    public:
        setValores(int, int);
    protected:
        int largura, altura;
};

class Retangulo : public Poligono{
    public:
        int area(){
            return largura*altura;
        }
};
```

Qualquer objeto que herdar de polígono, também terá esse recurso.

```
//representa poligonos que tem largura e altura
class Poligono{
    public:
        setValores(int, int);
    protected:
        int largura, altura;
};

class Retangulo : public Poligono{
    public:
        int area(){
            return largura*altura;
        }
};
```

Além das funções definidas para polígono, a classe Retangulo tem uma função área.

```
//representa poligonos que tem largura e altura
class Poligono{
    public:
        setValores(int, int);
    protected:
        int largura, altura;
};

class Retangulo : public Poligono{
    public:
        int area(){
            return largura*altura;
        }
};

class Triangulo : public Poligono{
    public:
        int area(){
            return (largura*altura)/2;
        }
};
```

```
//representa poligonos que tem largura e altura
class Poligono{
    public:
        setValores(int, int);
    protected:
        int largura, altura;
};

class Retangulo : public Poligono{
    public:
        int area(){
            return largura*altura;
        }
};

class Triangulo : public Poligono{
    public:
        int area(){
            return (largura*altura)/2;
        }
};
```

```
int main(){
    Retangulo ret;
    Triangulo tri;
    ret.setValores(7,3);
    tri.setValores(10,4);
    cout << ret.area() << endl;
    cout << tri.area() << endl;
}
```

```
//representa poligonos que tem largura e altura
class Poligono{
    public:
        setValores(int, int);
    protected:
        int largura, altura;
};

class Retangulo : public Poligono{
    public:
        int area(){
            return largura*altura;
        }
};

class Triangulo : public Poligono{
    public:
        int area(){
            return (largura*altura)/2;
        }
};
```

```
int main(){
    Retangulo ret;
    Triangulo tri;
    ret.setValores(7,3);
    tri.setValores(10,4);
    cout << ret.area() << endl;
    cout << tri.area() << endl;
}
```

```
//representa poligonos que tem largura e altura
class Poligono{
    public:
        setValores(int, int);
    protected:
        int largura, altura;
};

class Retangulo : public Poligono{
    public:
        int area(){
            return largura*altura;
        }
};

class Triangulo : public Poligono{
    public:
        int area(){
            return (largura*altura)/2;
        }
};
```

```
int main(){
    Retangulo ret;
    Triangulo tri;
    ret.setValores(7,3);
    tri.setValores(10,4);
    cout << ret.area() << endl;
    cout << tri.area() << endl;
}
```