```
Razionale r (2,3);

Razionale s;

s = r;
```

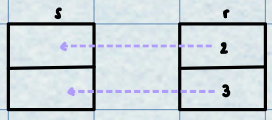
operatore di assegnamento ma non ancora sovraccaricato

s.operator = (r)

- comportamento di default dell'operatore fra oggetti della stessa classe

  newObj = oldObj;

- copia attributo per attributo tutti gli attributi di oldObj a newObj



Assegnamento operator = di default

```
const C& operator= (const C& v) {
        if (this != &v) {
                //copia attributo per attributo
                }
        return *this;
        }
```

infine si ritorna l'oggetto assegnato

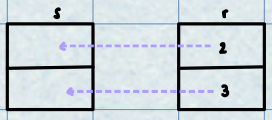
nella classe Stack

```
const Stack & operator = (const Stack& other) {
    if (this != &other) {
        delete [] this->A;
        A = new int [other.dim];
        for (int i= 0; i < other.top; i++){
            A[i]= other.A[i];}
        top = other.top;
        dim = other.dim;}
        return *this;}
```

Costruttore di copia

Di default, ogni classe ha un costruttore di copia, con un singolo parametro dello stesso tipo della classe

```
Razionale r(2,3);
Razionale s(r);
```



copia attributo per attributo

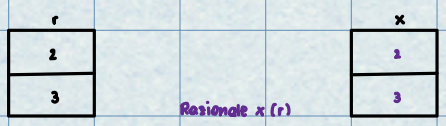
copia attributo per attributo ogni attributo dell'oggetto passato come parametro nell'oggetto dichiarato

- Il costruttore di copia viene chiamato automaticamente durante il passaggio di parametri e risultati per valore

```
int main() {
    Razionale r(2,3);
    ...
    f(r);
    ...
}
```

```
void f(Razionale x) {
    ...
}
```

passaggio per valore



Razionale x(r)

Come per l'operatore di assegnamento, il comportamento di default del costruttore di copia potrebbe non andare bene in presenza di allocazione dinamica della memoria.

Il costruttore di copia di una classe C è

- public
- un costruttore
- un unico parametro di tipo const C& (const reference)

```
C(const C& c) {
    ...
}
```

nella classe Stack

```
Stack(const C& other) {
    A = new int[other.dim];
    for(int i=0; i < other.top ; i++){
        A[i] = other.A[i];}
```

```
        top= other.top;
        dim= other.dim;
        }
```

The Rule of three

"Se classe definisce esplicitamente almeno uno tra distruttore, costruttore di copia e operatore di assegnamento, allora tutti e tre i metodi devono essere definiti esplicitamente"

Liste di interi

```
#include <iostream>
using namespace std;
struct elem {       // datatype dell'elemento della lista concatenata
        int data;
        elem* next;
        }
class int_list{
        private:
                elem * l;       //nextatore di inizio della lista concatenata
                int n;          // n° elementi della lista
        public:
                int_list(){             // costruttore senza parametri
                        l = NULL;
                        n = 0; }
                ~int_list(){
                        if ( l != NULL){
                                node * prev; }
                        do {
```

```
        prev = l;

        l = l->next;

        delete prev;

        } while (l != NULL)

}

]

void  add_first (int x) {

        elem* t = new elem;

        t->data = x;

        t->next = l;  // N.B.  campo "l" dell'oggetto d'invocazione

        l = t;

        n++;

        return; }

int  remove_first () {

        if (l == NULL) {

                throw string ("empty list "); }

        int x;

        x= l->data;

        elem* t = l;

        l = l->next;

        delete t;

        n--;

        return x; }

void  remove_element (int x) {          //rimuove  dalla lista il primo elemento con valore x
```

```
        if (l->data == x) {          // la lista rimane inalterata se x non è presente
            remove_first();
            return; }
        elem * t = l;
        while (t->next != nullptr) {
            if ( t->next->data ==x ) {
                elem *  next = t->next->next;
                delete  t->next;
                t->next = next;
                break; }
            else {
                t= t->next; }
        return; }

int size () const {
        return n; }

bool contains (int x) const {
```