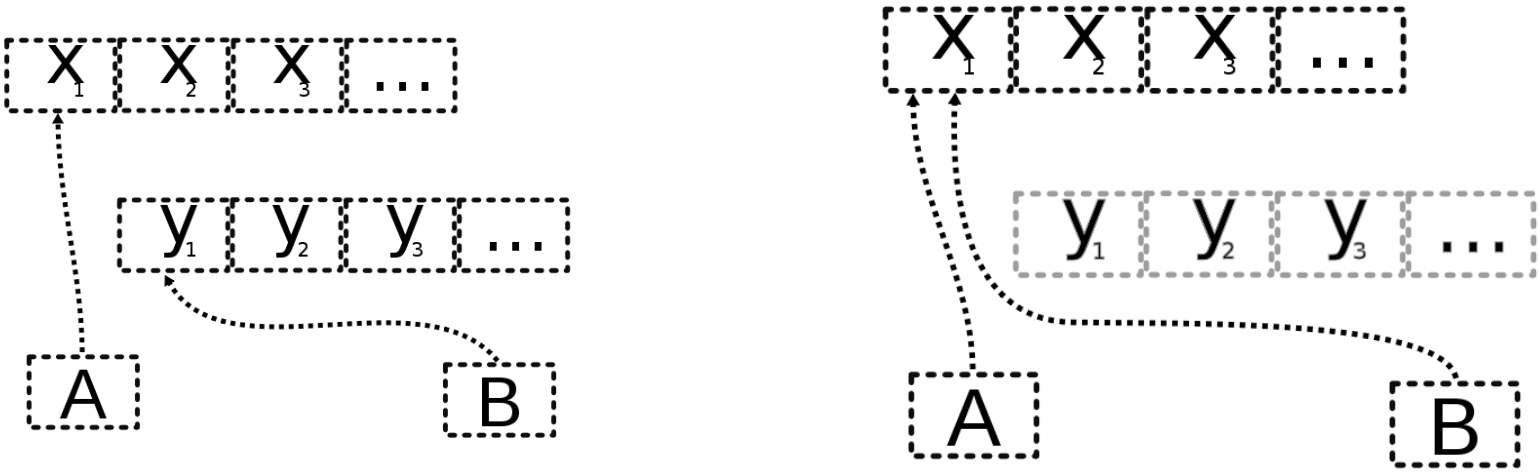
Copying

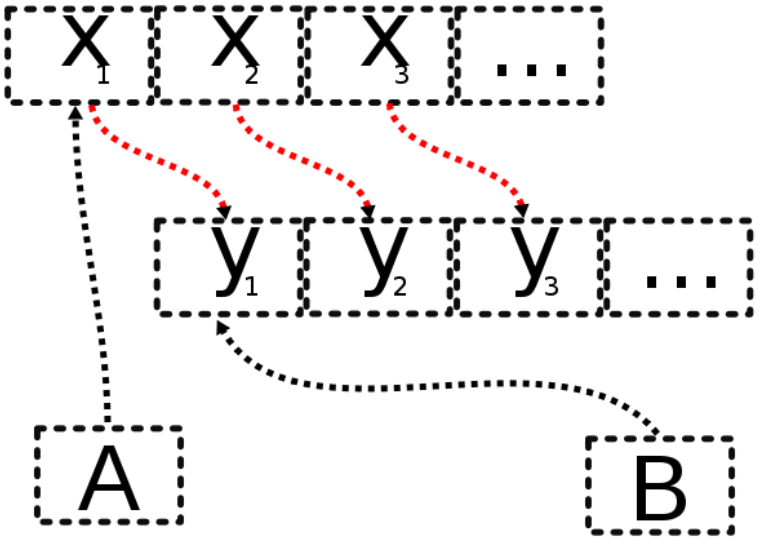
Thomas van Bruggen
Division of medical and biological Informatics

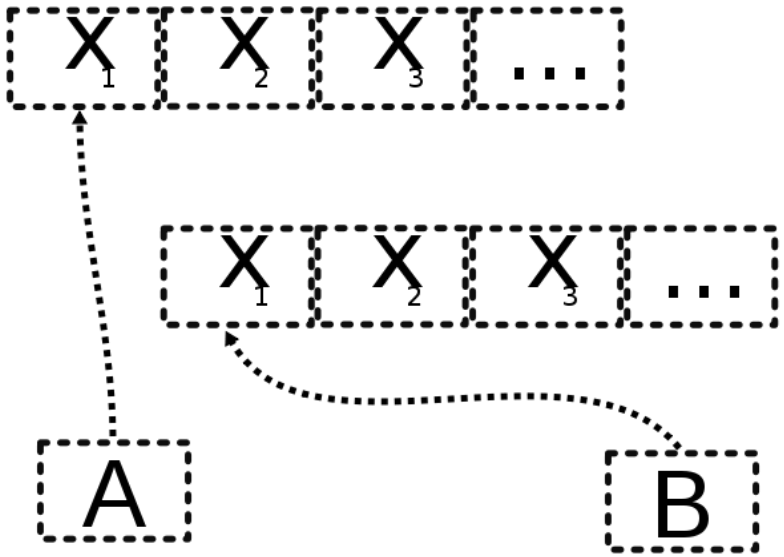
Shallow Copy



- Memory shared. Modifying the one will also modify the other.
- Memory leak in Y1…..Yn

Deep Copy





Example shallow copy

```
#include <iostream>
class base {
public:
  int i;
  base(int j=0) { i=j; }
};
```

```
base *p1=new base(23);
base *p2;
p2=p1;
cout<<"\naddress of P1:"<<p1;
cout<<"\nvalue at p1:"<<p1->i;
cout<<"\naddress of P2:"<<p2;
cout<<"\nvalue at p2:"<<p2->i;

delete p2;

cout<<"\naddress of P1 after delete:"<<p1;
cout<<"\naddress of P2 after delete:"<<p2;
cout<<"\nvalue in P1 after delete:"<<p1->i
cout<<"\nvalue in P2 after delete:"<<p2->i
```

```
address of P1:0xa38010
value of P1:23
address of P2:0xa38010
value of P2:23

address of P1 after delete:0xa38010
address of P2 after delete:0xa38010
value in P1 after delete:0
value in P2 after delete:0
```

Example deep copy

```
base o1(67);
base o2;
o2=o1;        //contents are copied. But, the addresses remained different
cout << "value in o1:"<< o1.i;
cout << "value in o2 after copy:“ << o2.i;
return 0;
```

```
value in o1:67
value in o2 after copy:67
```

Copy constructors and assignment operators

- C++ is designed so that user-defined types function like primitive types.
- Allows to define a copy constructor and an assignment operator.
- Provides default CC and AO that perform a memberwise copy (shallow copying).

```
Vector v1;          // v1 will be constructed by the standard constructor
Vector v2(v1);      // The copy constructor will be used to create a copy of v1
Vector v3 = v1;     // v3 will typically be constructed by the copy constructor
Vector v4;
v4 = v1;            // The assignment operator will be used here
```

- All is well as long as classes do not use dynamically allocated memory (pointers)

Problem

```
Class MyString
{
private:
  char *m_String;
  int m_Length
public:
  MyString(char *str=“”){
    int m_Length = strlen(str) + 1;
    m_String = new char[m_Length];
    strncpy(m_String, str, m_Length);
  }
  ~MyString(){
      delete[] m_String;
      m_String = 0;
    }
  char* GetString(){return m_String}
```

```
MyString cHello(“Hello world!”)
If(true){
  MyString cCopy = cHello;
} // cCopy goes out of scope

std::cout << c.Hello.GetString() << std::endl;
```

Sollution: Deep copying copy constructor

```
MyString::MyString(const MyString& cSource){
if(cSource.m_String) {
    int length = cSource.m_Length();
    m_String = new char[length];
    strncpy(m_String, cSource.m_String(), length);
  }
  else
    m_pchString = 0;
}
```

Deep copy assignment operator

```
MyString& MyString::operator=(const MyString& cSource)
{
  if(this == &cSource)
    return *this;
  delete[] m_pchString;
  m_nLength = cSource.m_Length;

  if(cSource.m_nLength;
  if(cSource.m_pchString)
  {
    m_pchString = new char[m_nLength];
    strncpy(m_pchString, cSource.m_pchString, m_nLength);
  }
  else
    m_pchString = 0;
  return *this
}
```

Differences with copy constructor

- Return *this so we can 'chain' them
- Explicit deallocations
- Self-assignment check

What happens if we do this without checking For self-assignment?

```
cHello = cHello;
```

Final remarks

- It is possible to prevent copying by overriding the copy constructor and assignment operator as private.
- Recently mitk::BaseData derivatives got the possibility of having a Clone method that calls a copy constructor (to be implemented by the user).

- http://www.learncpp.com/cpp-tutorial/912-shallow-vs-deep-copying/
- http://en.wikipedia.org/wiki/Object_copy#Another_example_of_deep_and_shallow_copying_in_C.2B.2B