CSI33 Data Structures

Department of Mathematics and Computer Science
Bronx Community College

November 1, 2017

Outline

Outline

Storing Variables In Python and C++

Storing Variables In Python

- Values of Python variables (objects) are found by references (addresses) associated with the variable names.
- The memory used to store the value (the object) is allocated when it is needed.
- The object's information includes its type and a reference count.
- Assignment in Python changes the reference, not the object itself.

Storing Variables In Python and C++

Storing Variables In C++

- Values of C++ variables are in locations directly associated with the variable names.
- The memory used to store declared variables must be allocated at compile time.
- No reference counting is used; the variable's memory is deallocated when the program leaves its scope.
- Assignment in C++ changes the variable's value itself.

Storing Variables In Python and C++

Objects In Python

- Assignment of a variable does not change the object it refers to.
- The memory used to store the value (the object) is created when it is needed.

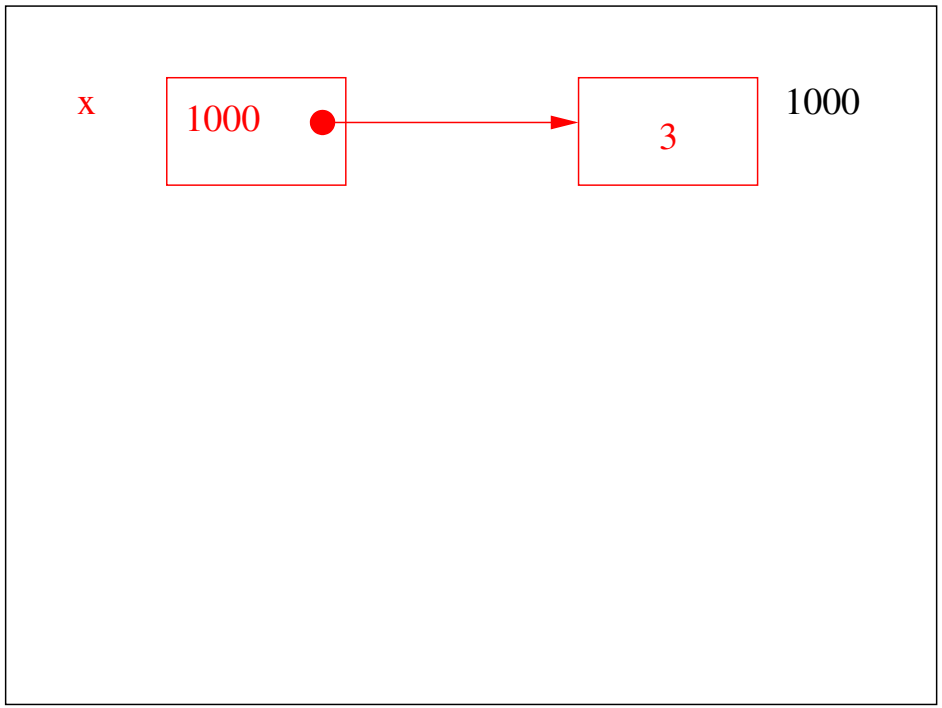
Storing Variables In Python and C++

Objects In C++

- Assignment of a variable change the attributes of the object in the variable's location. (The assignment operator = can be overloaded in C++. The default behavior is to assign the attribute values of the object on the right side to those of the object on the left.
- The memory used to store the value (the object) is created when the program enters its scope.

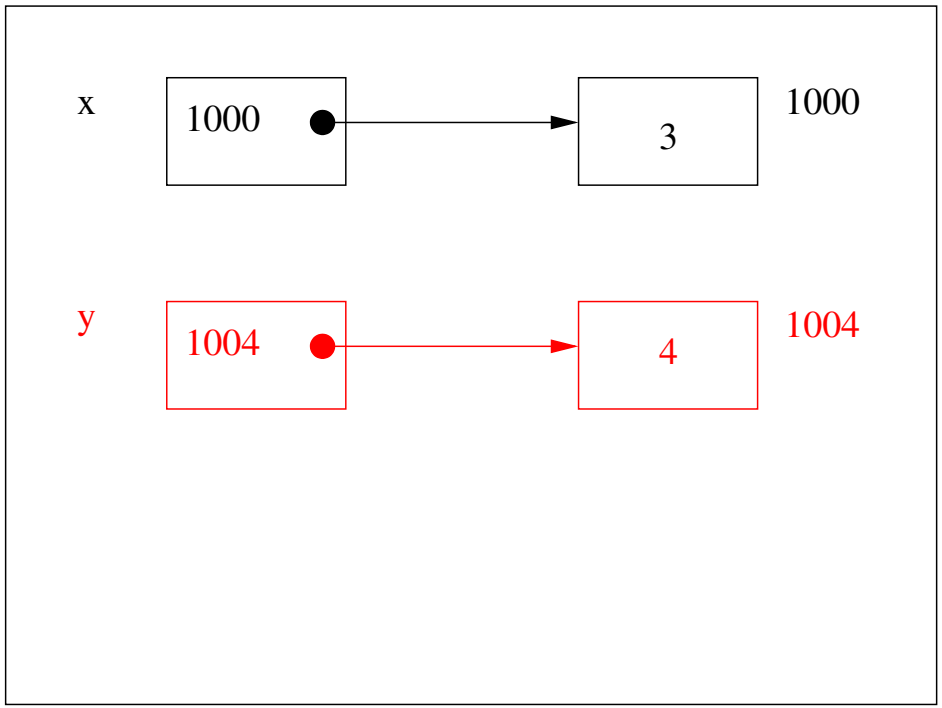
Python Memory Example

```
x = 3
y = 4
z = x
x = y
```



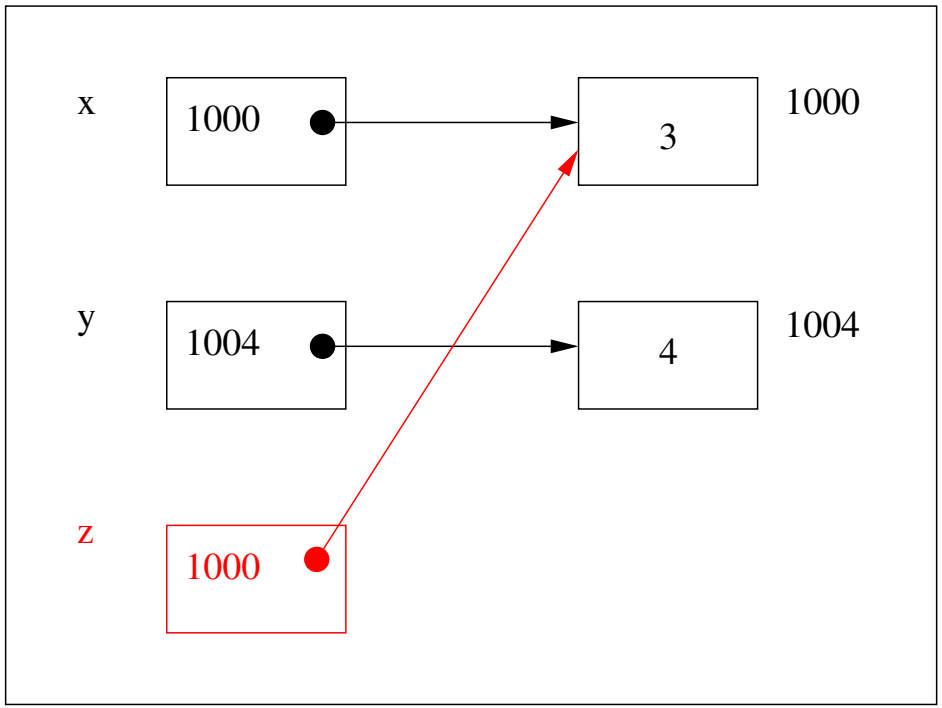
Python Memory Example

```
x = 3
y = 4
z = x
x = y
```



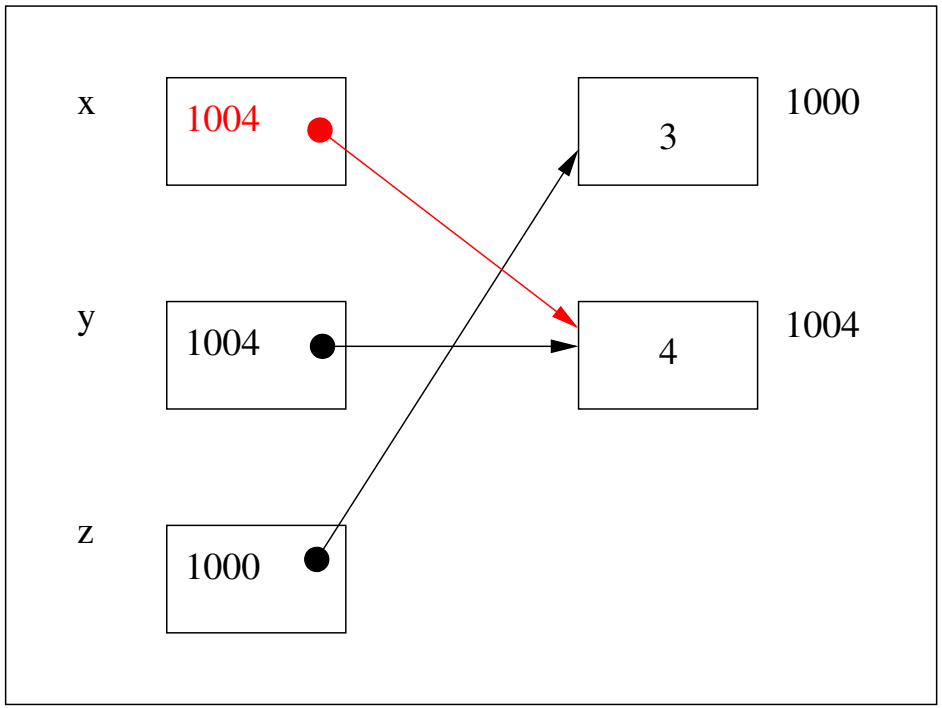
Python Memory Example

```
x = 3
y = 4
z = x
x = y
```



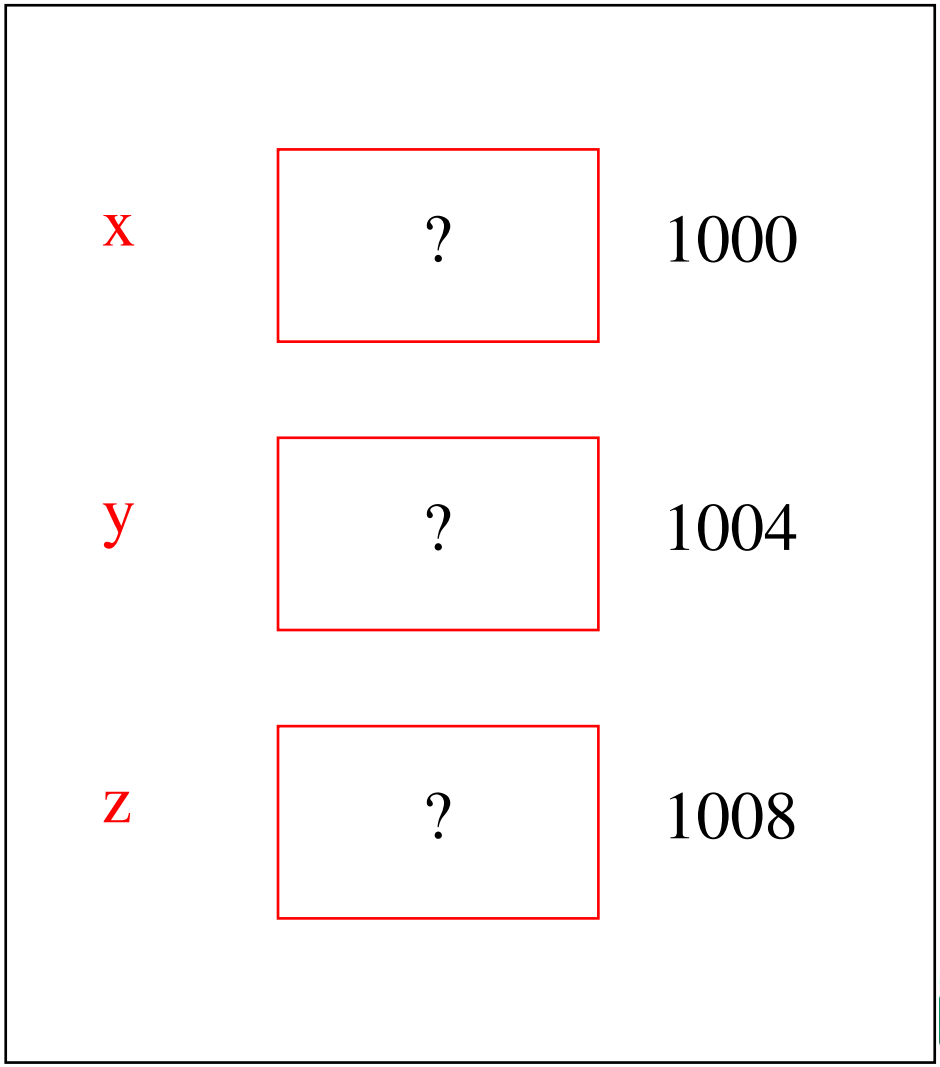
Python Memory Example

```
x = 3
y = 4
z = x
x = y
```



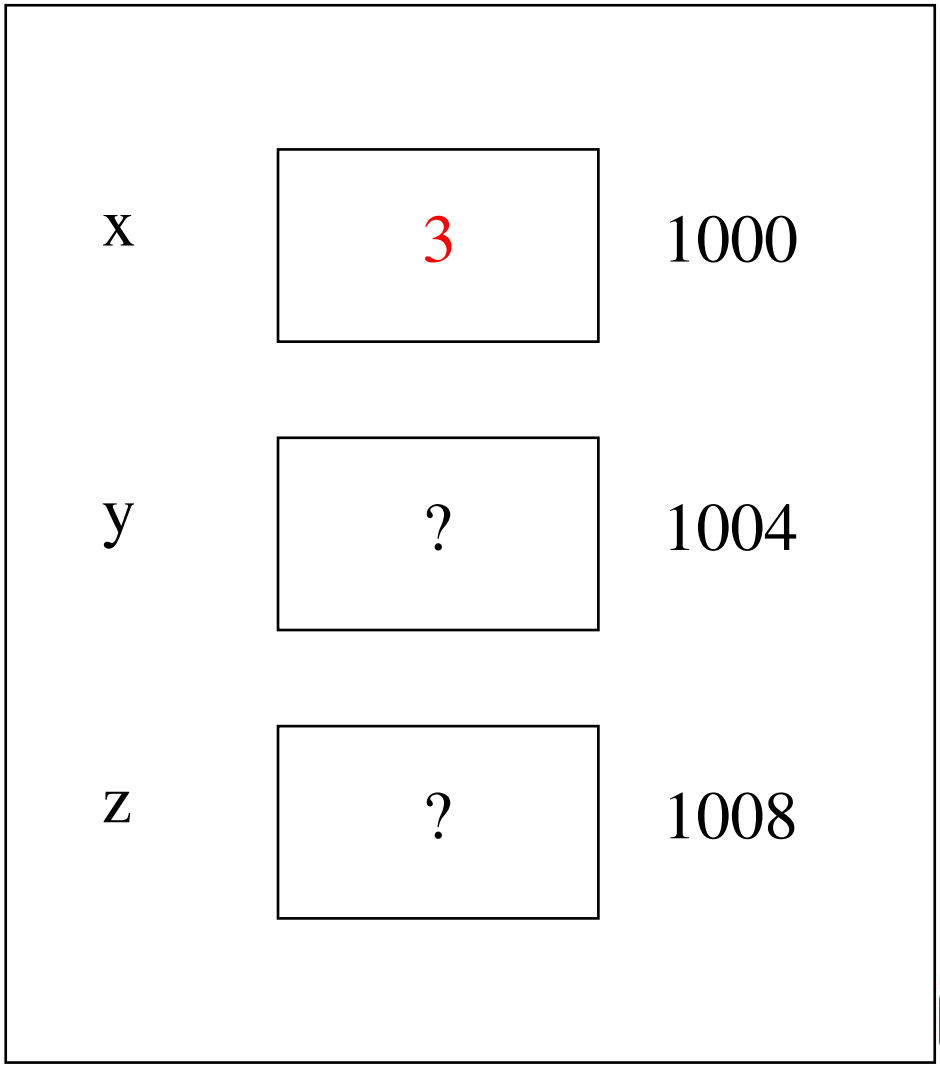
C++ Memory Example

```
int x, y, z;
x = 3;
y = 4;
z = x;
x = y;
```



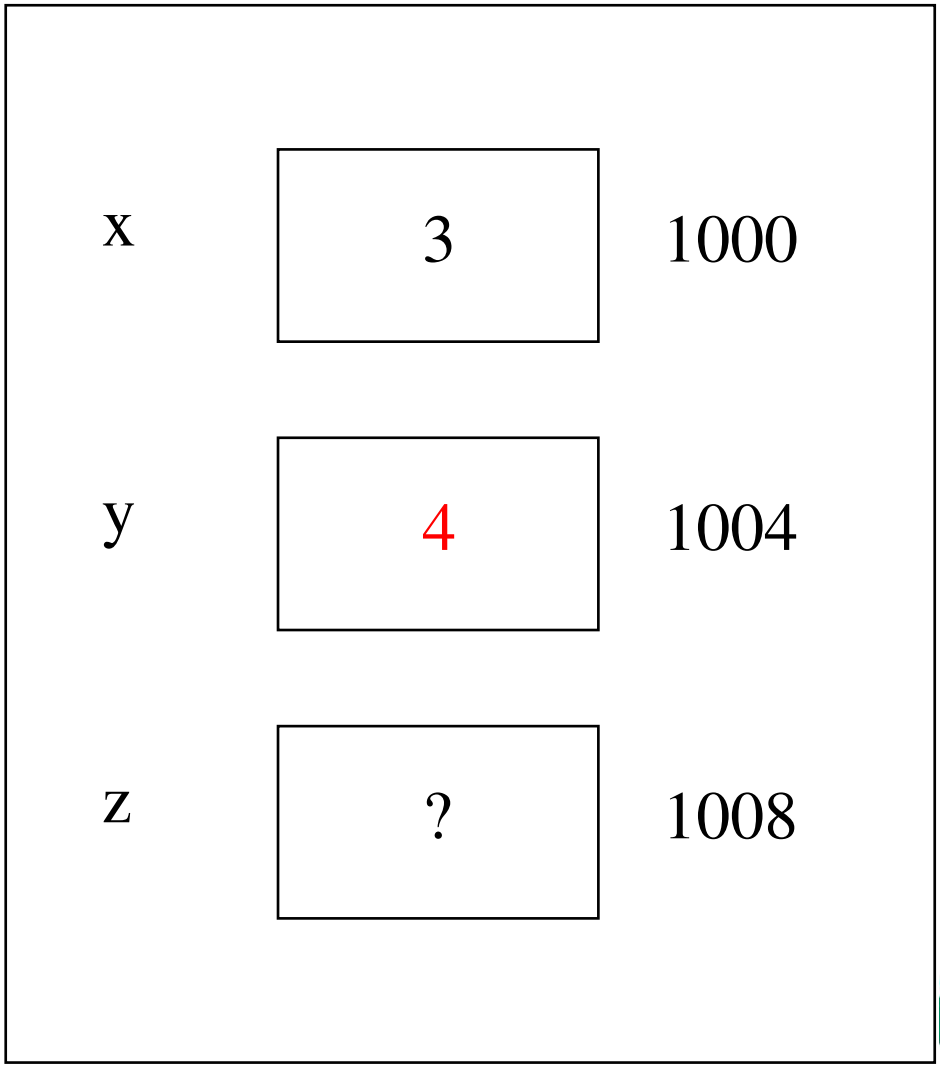
C++ Memory Example

```
int x, y, z;
x = 3;
y = 4;
z = x;
x = y;
```



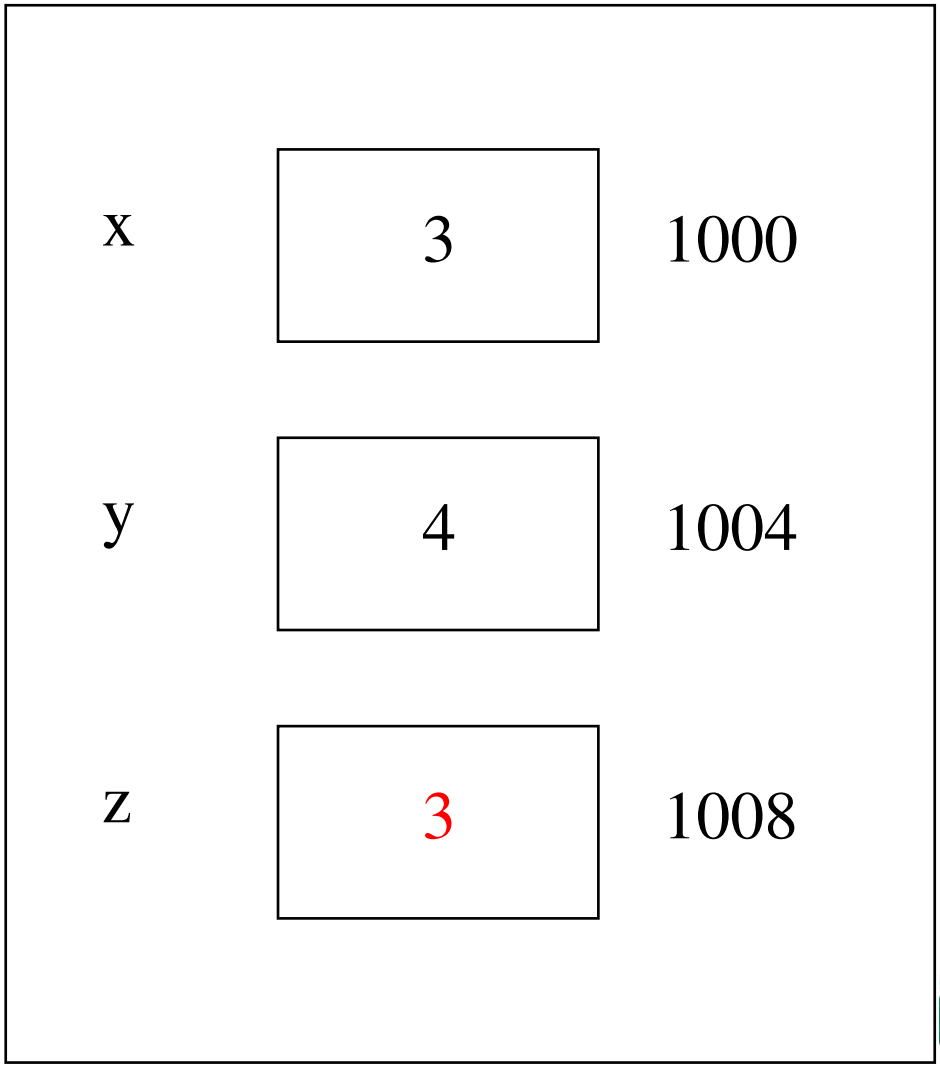
C++ Memory Example

```
int x, y, z;
x = 3;
y = 4;
z = x;
x = y;
```



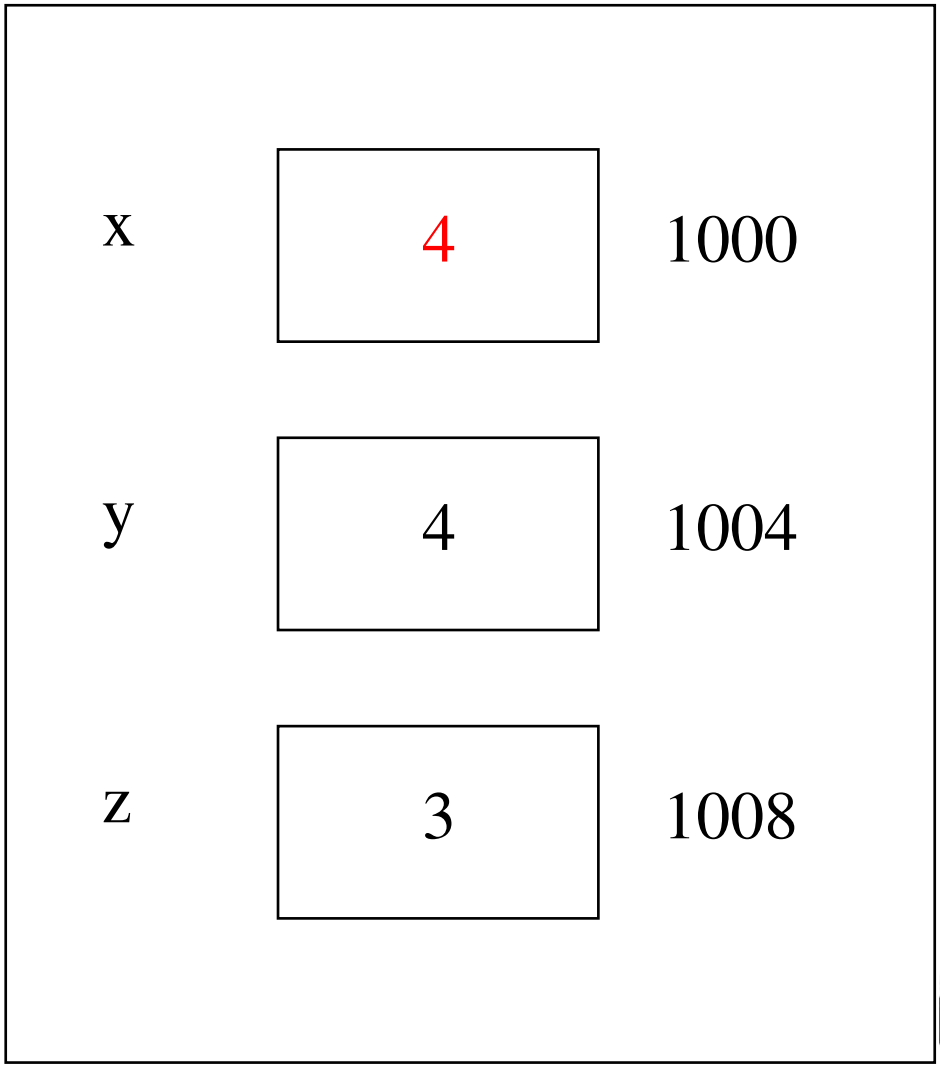
C++ Memory Example

```
int x, y, z;
x = 3;
y = 4;
z = x;
x = y;
```



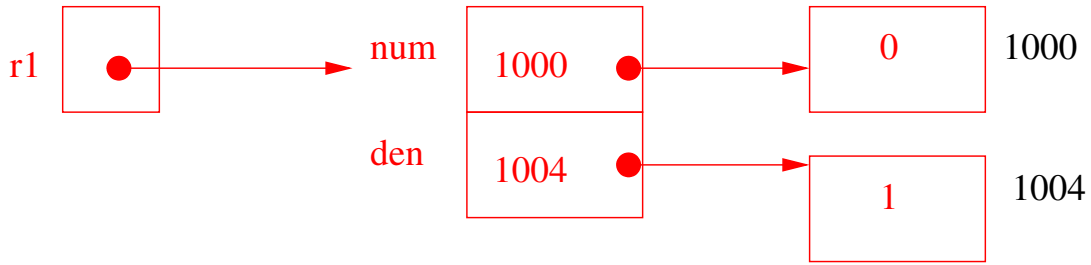
C++ Memory Example

```
int x, y, z;
x = 3;
y = 4;
z = x;
x = y;
```



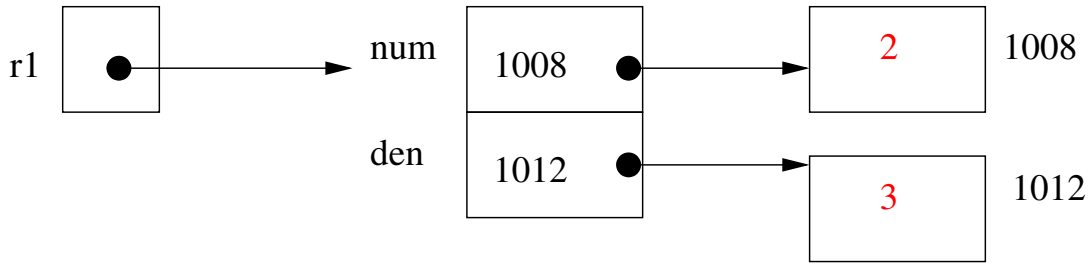
Python Object Example

```
r1 = Rational()
r1.set(2, 3)
r2 = r1
r1.set(1, 3)
```



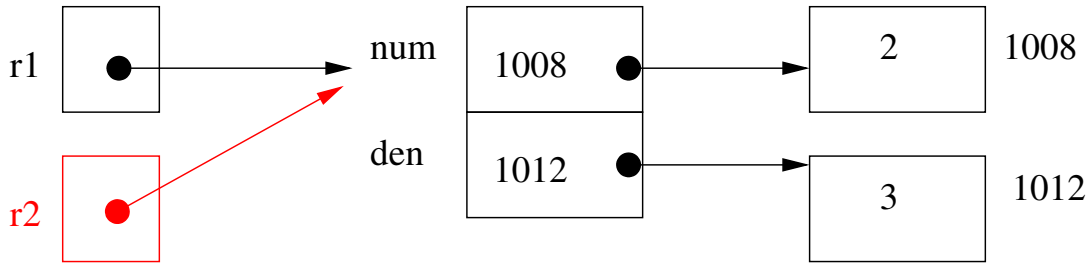
Python Object Example

```
r1 = Rational()
r1.set(2, 3)
r2 = r1
r1.set(1, 3)
```



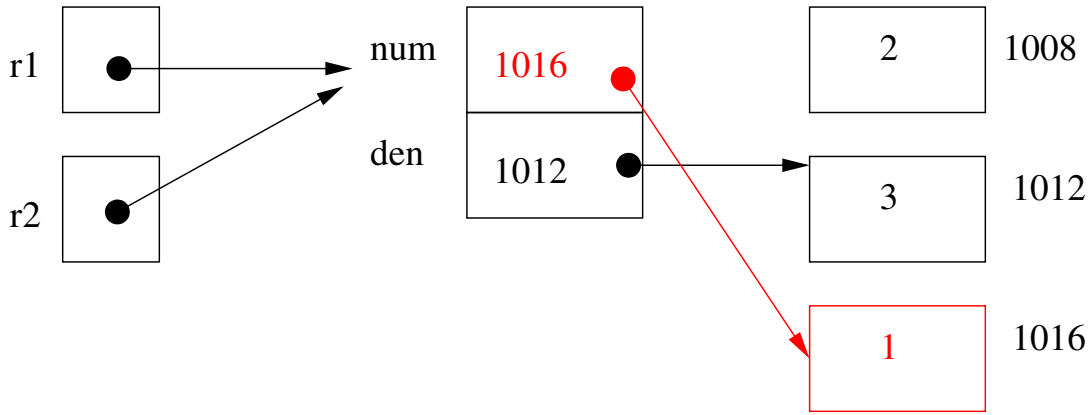
Python Object Example

```
r1 = Rational()
r1.set(2, 3)
r2 = r1
r1.set(1, 3)
```



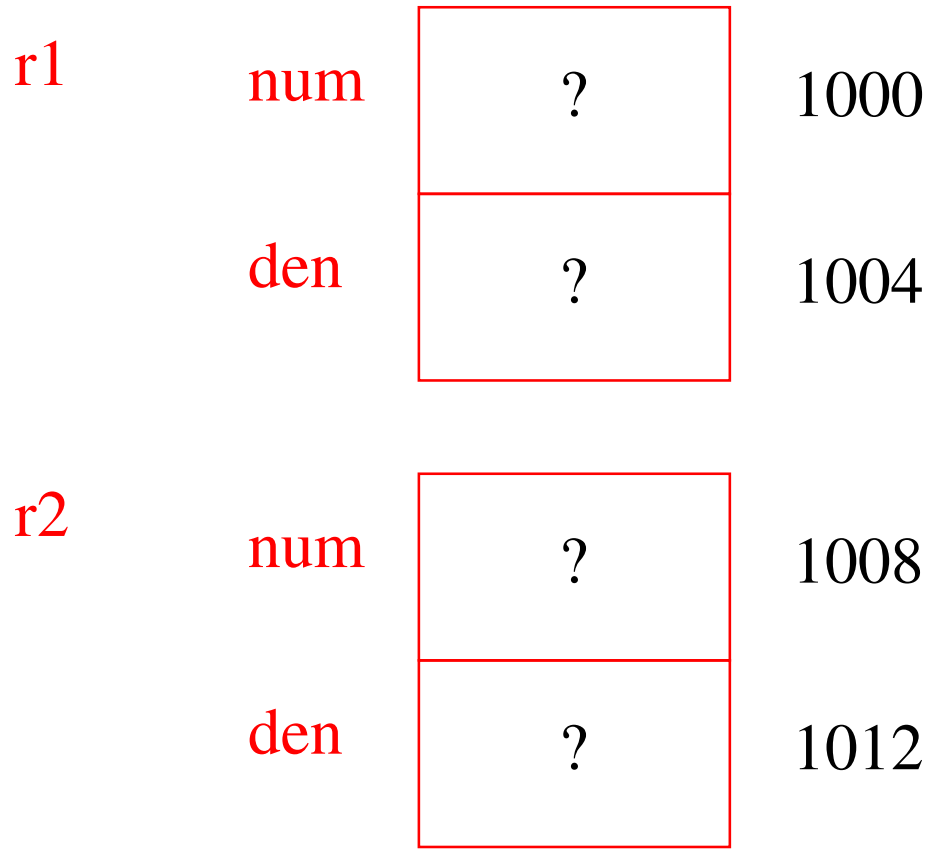
Python Object Example

```
r1 = Rational()
r1.set(2, 3)
r2 = r1
r1.set(1, 3)
```



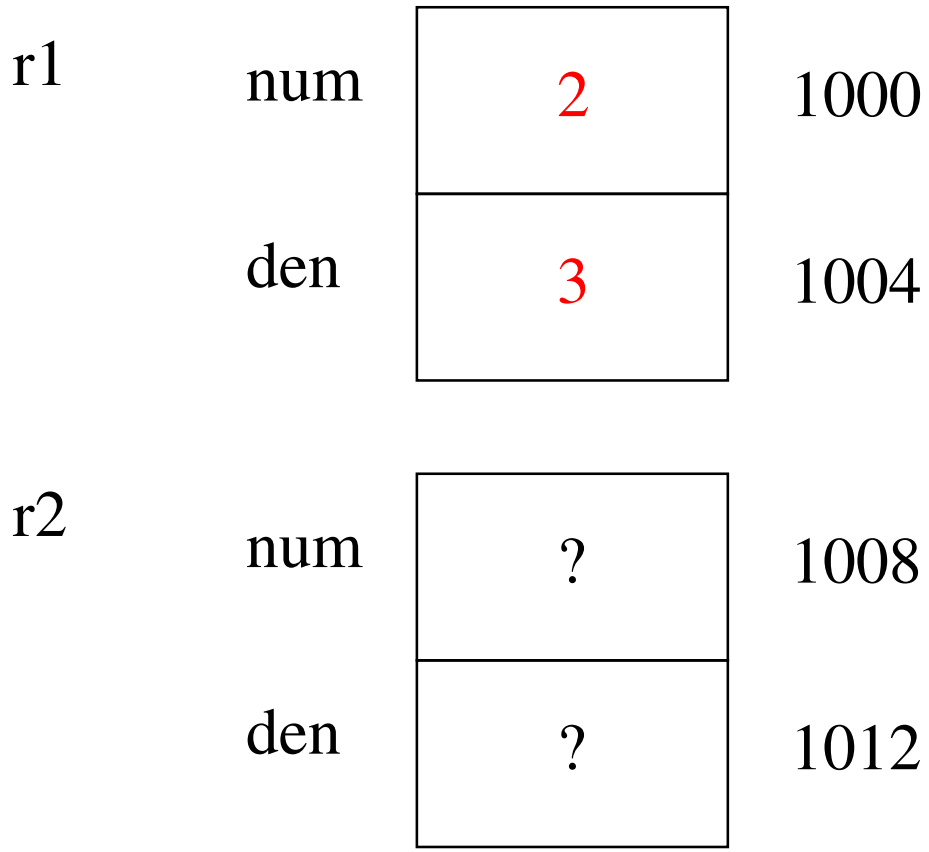
C++ Object Example

```
Rational r1, r2;
r1.set(2, 3);
r2 = r1;
r1.set(1, 3);
```



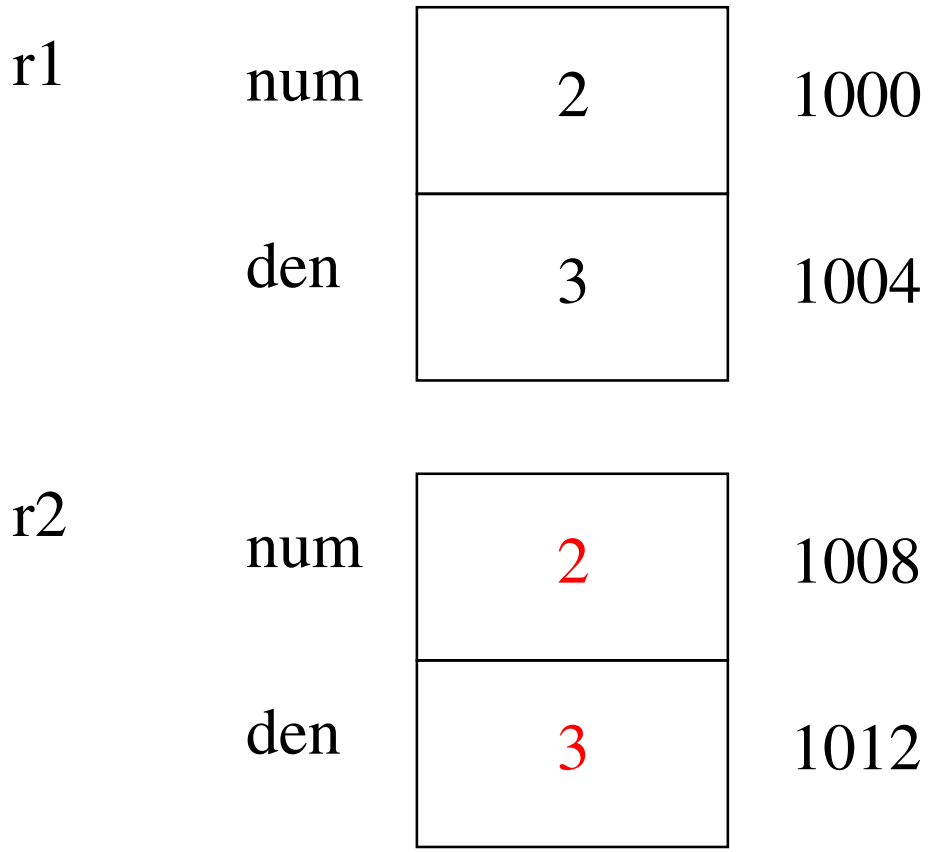
C++ Object Example

```
Rational r1, r2;
r1.set(2, 3);
r2 = r1;
r1.set(1, 3);
```



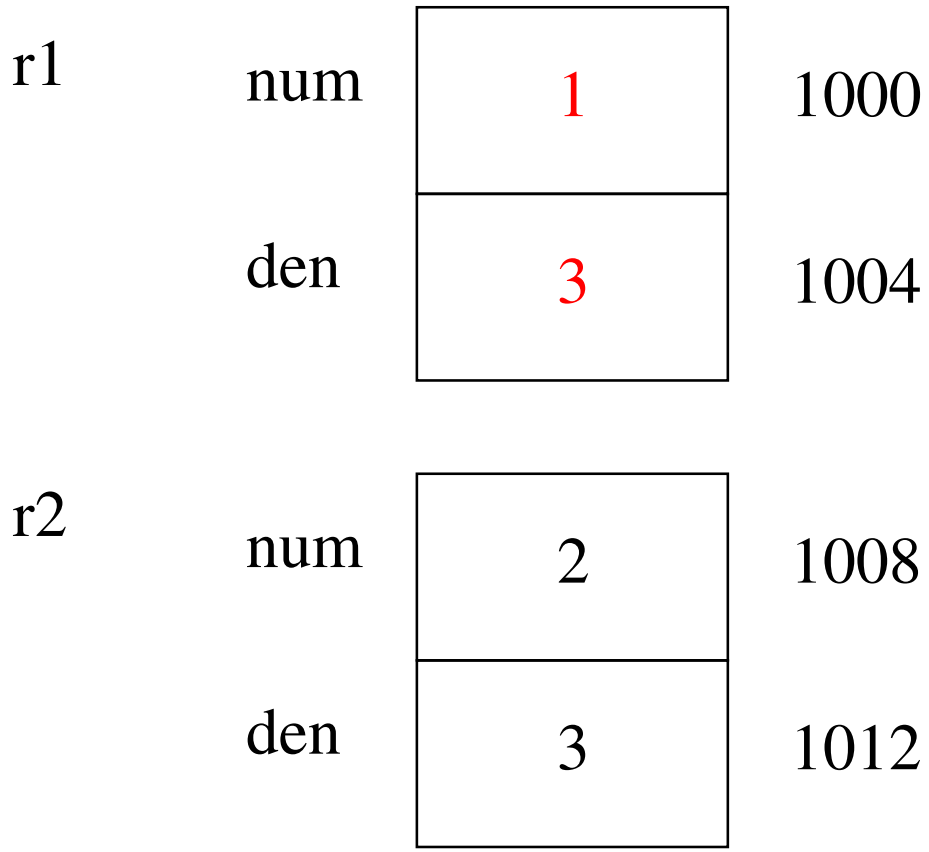
C++ Object Example

```
Rational r1, r2;
r1.set(2, 3);
r2 = r1;
r1.set(1, 3);
```



C++ Object Example

```
Rational r1, r2;
r1.set(2, 3);
r2 = r1;
r1.set(1, 3);
```



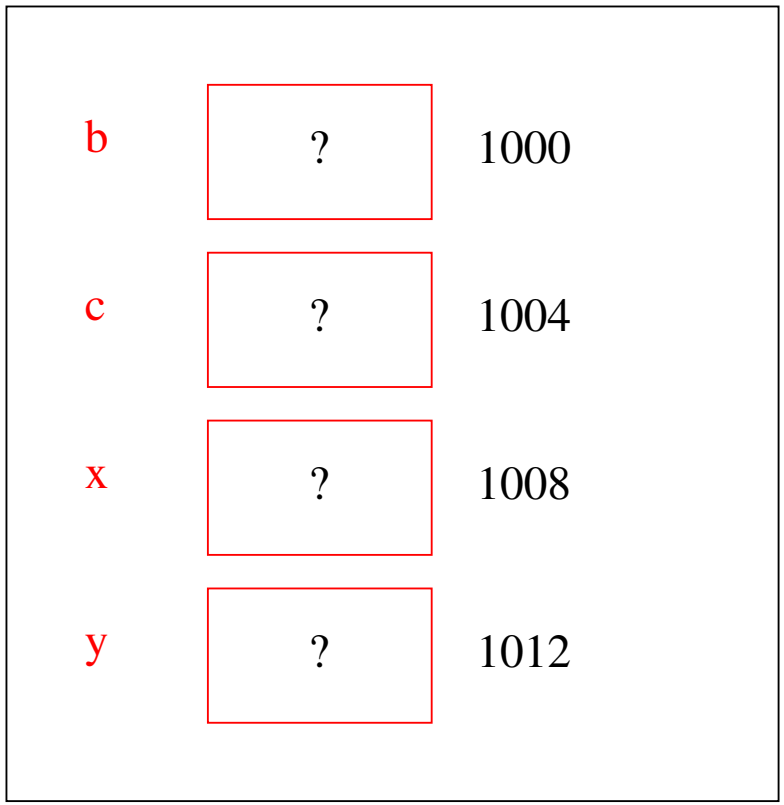
Pointer Syntax

Pointer Declaration

- A * (the dereference operator) prefixes the variable name and follows the type. It gives the **value** pointed to by the pointer variable after it.
- A pointer declaration reserves space for the address of an object of the type given before the * symbol. (A C++ pointer plays the role of a reference in the Python style.)
- The ampersand (&), or reference operator, is the opposite of *. It gives the **address** of the variable after it.

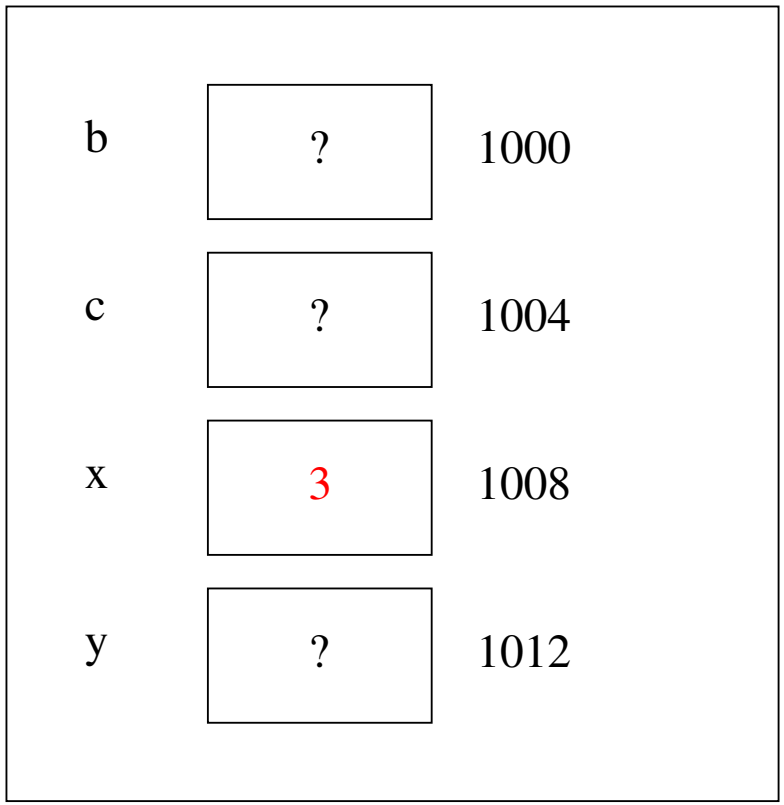
C++ Pointer Example 1

```
int *b, *c, x, y;
x = 3;
y = 5;
b = &x;
c = &y;
*b = 4;
*c = *b + *c;
c = b;
*c = 2;
```



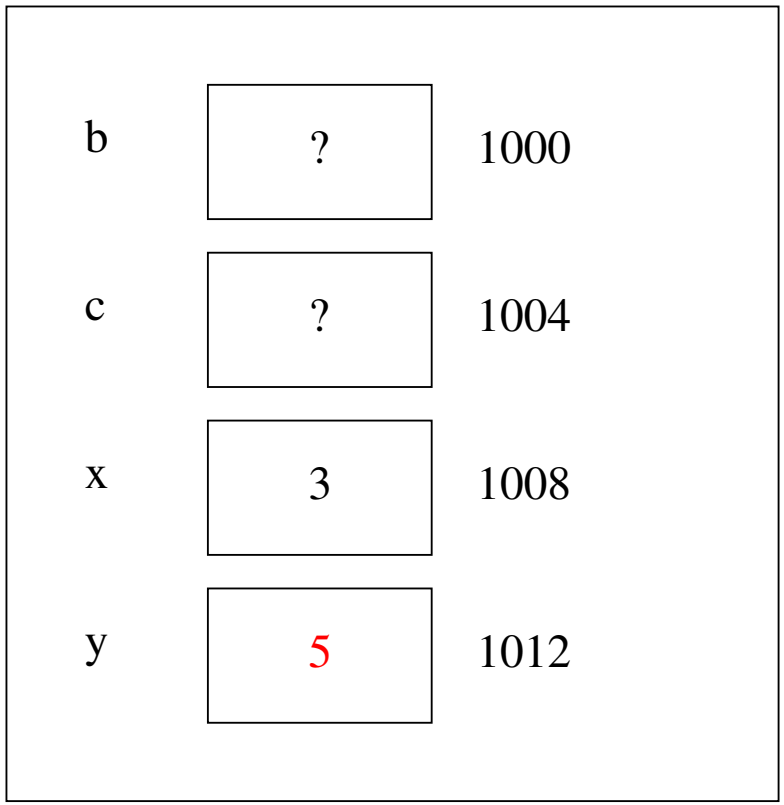
C++ Pointer Example 1

```
int *b, *c, x, y;
x = 3;
y = 5;
b = &x;
c = &y;
*b = 4;
*c = *b + *c;
c = b;
*c = 2;
```



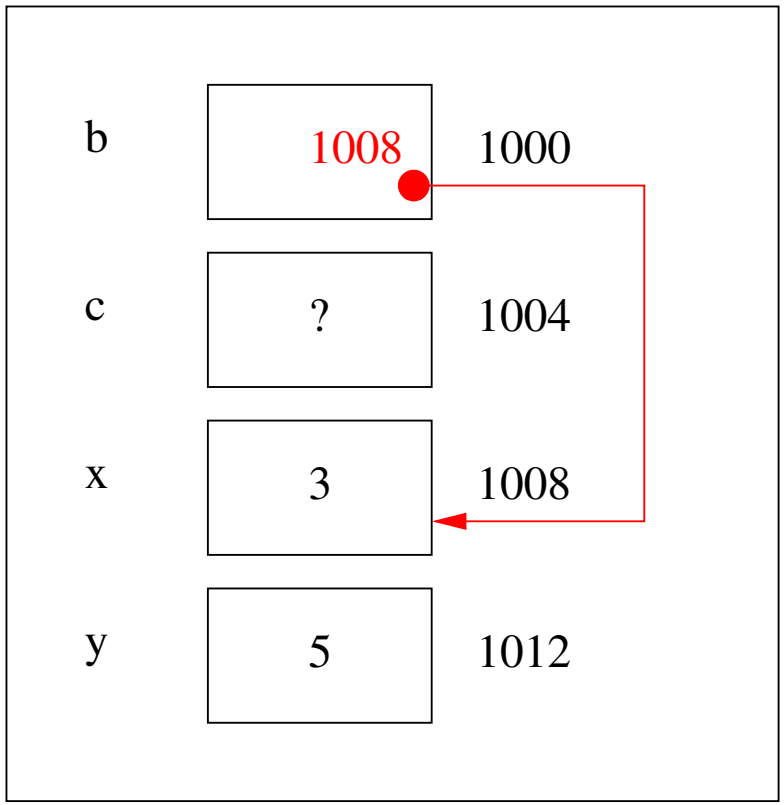
C++ Pointer Example 1

```
int *b, *c, x, y;
x = 3;
y = 5;
b = &x;
c = &y;
*b = 4;
*c = *b + *c;
c = b;
*c = 2;
```



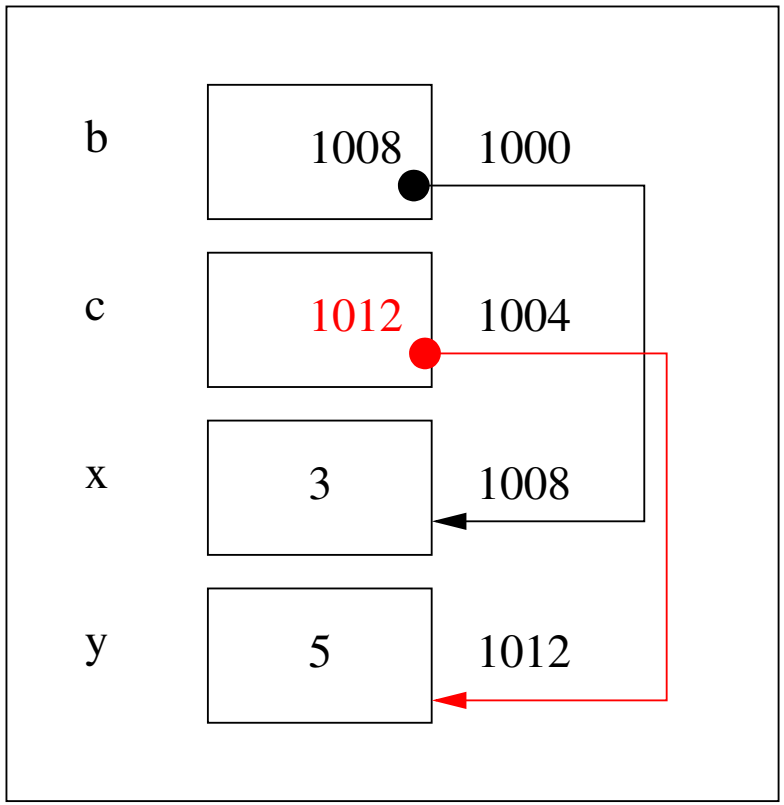
C++ Pointer Example 1

```
int *b, *c, x, y;
x = 3;
y = 5;
b = &x;
c = &y;
*b = 4;
*c = *b + *c;
c = b;
*c = 2;
```



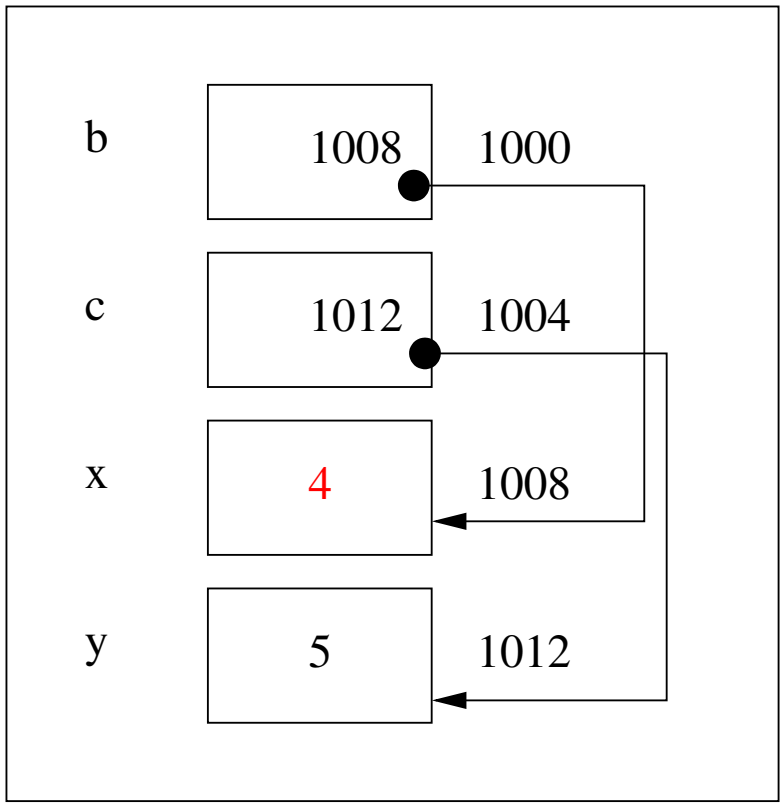
C++ Pointer Example 1

```
int *b, *c, x, y;
x = 3;
y = 5;
b = &x;
c = &y;
*b = 4;
*c = *b + *c;
c = b;
*c = 2;
```



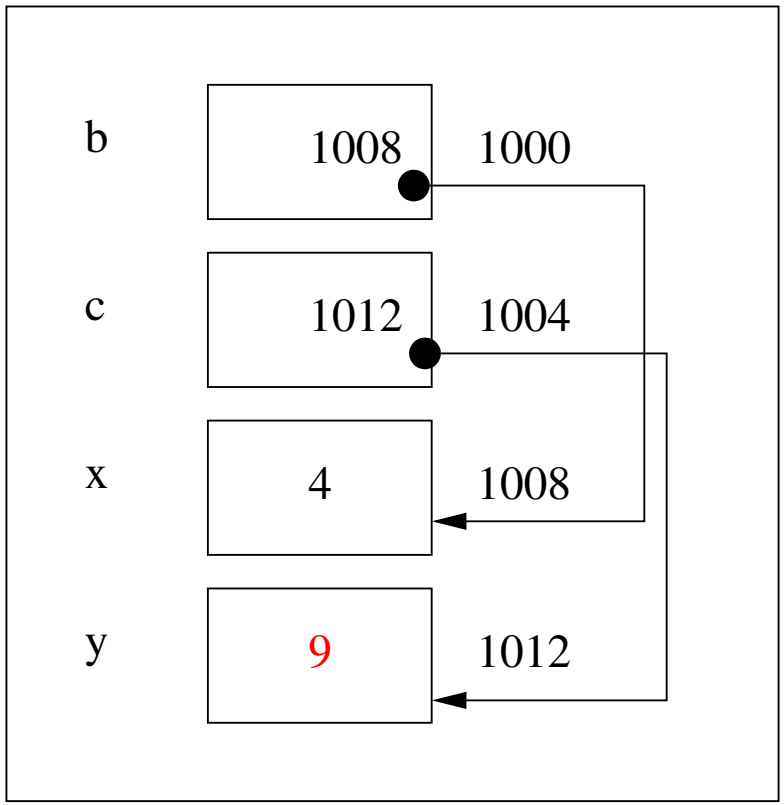
C++ Pointer Example 1

```
int *b, *c, x, y;
x = 3;
y = 5;
b = &x;
c = &y;
*b = 4;
*c = *b + *c;
c = b;
*c = 2;
```



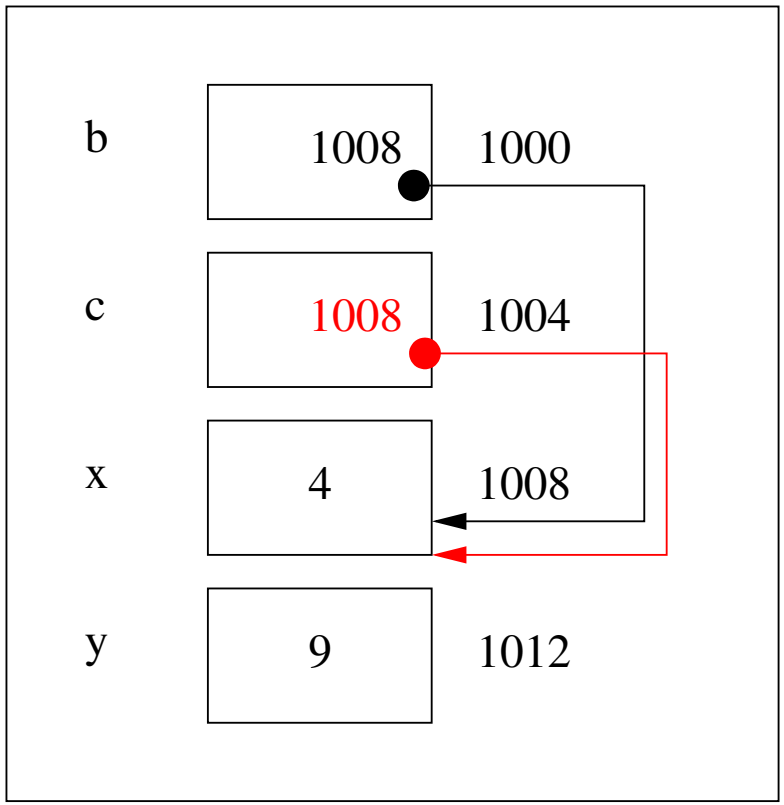
C++ Pointer Example 1

```
int *b, *c, x, y;
x = 3;
y = 5;
b = &x;
c = &y;
*b = 4;
*c = *b + *c;
c = b;
*c = 2;
```



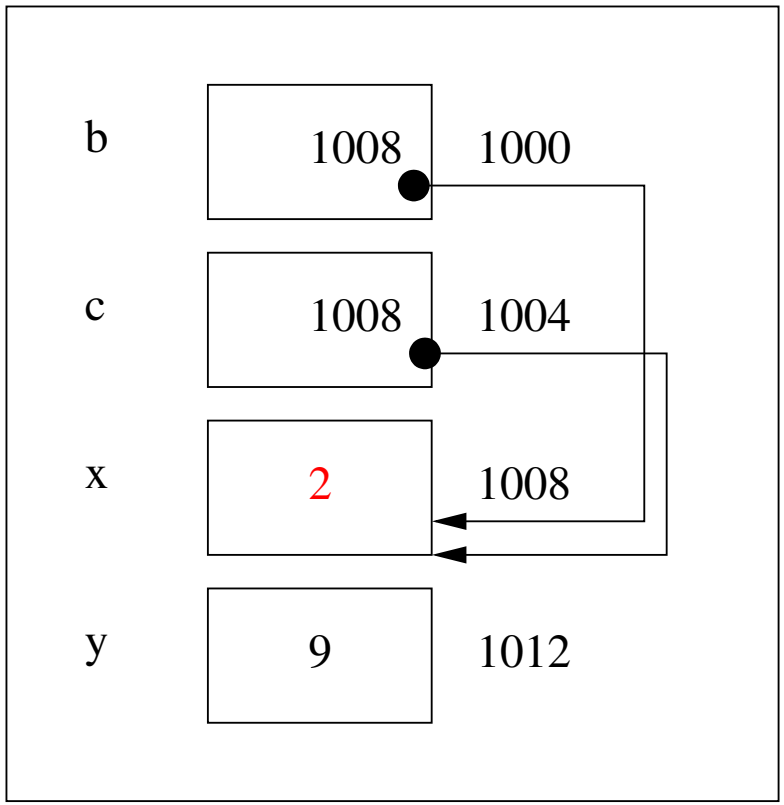
C++ Pointer Example 1

```
int *b, *c, x, y;
x = 3;
y = 5;
b = &x;
c = &y;
*b = 4;
*c = *b + *c;
c = b;
*c = 2;
```



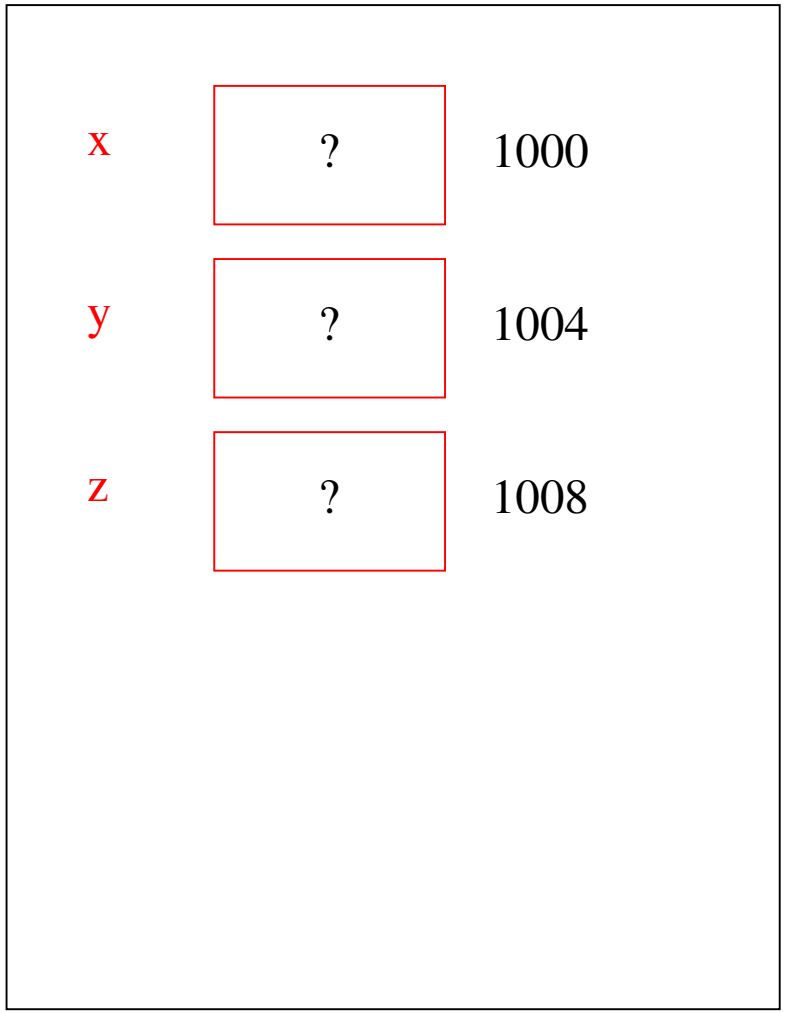
C++ Pointer Example 1

```
int *b, *c, x, y;
x = 3;
y = 5;
b = &x;
c = &y;
*b = 4;
*c = *b + *c;
c = b;
*c = 2;
```



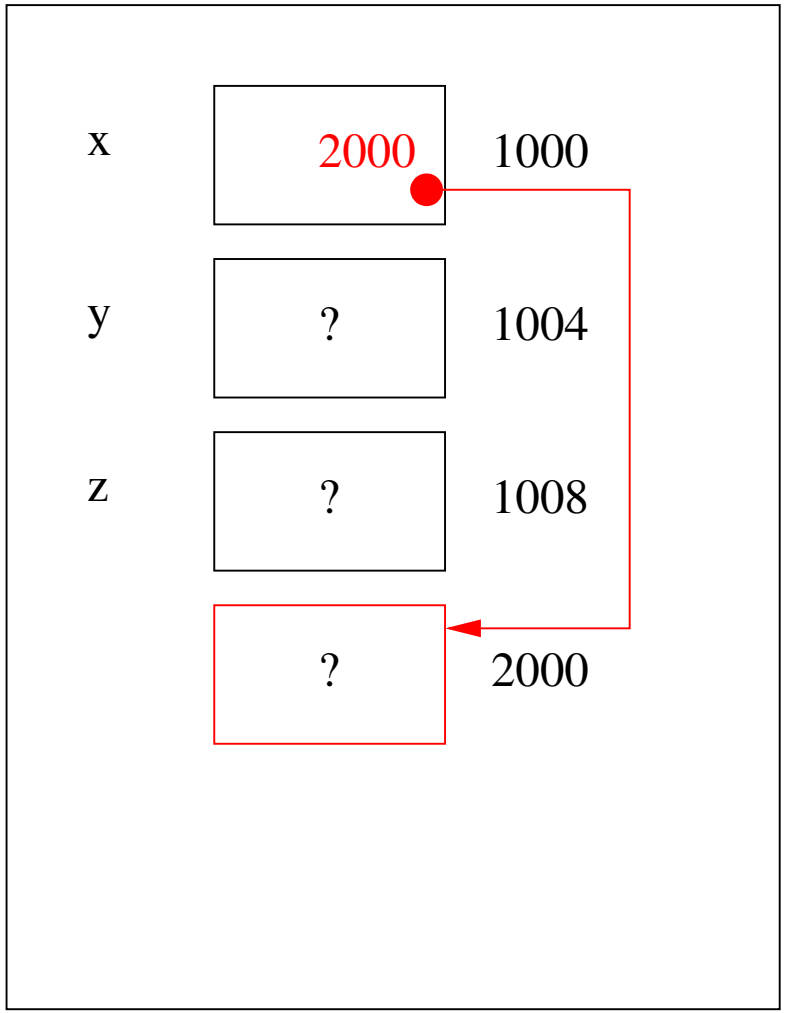
C++ Pointer Example 2

```
int *x, *y, *z;
x = new int;
*x = 3;
y = new int;
*y = 4;
z = x;
x = y;
delete z;
delete y;
```



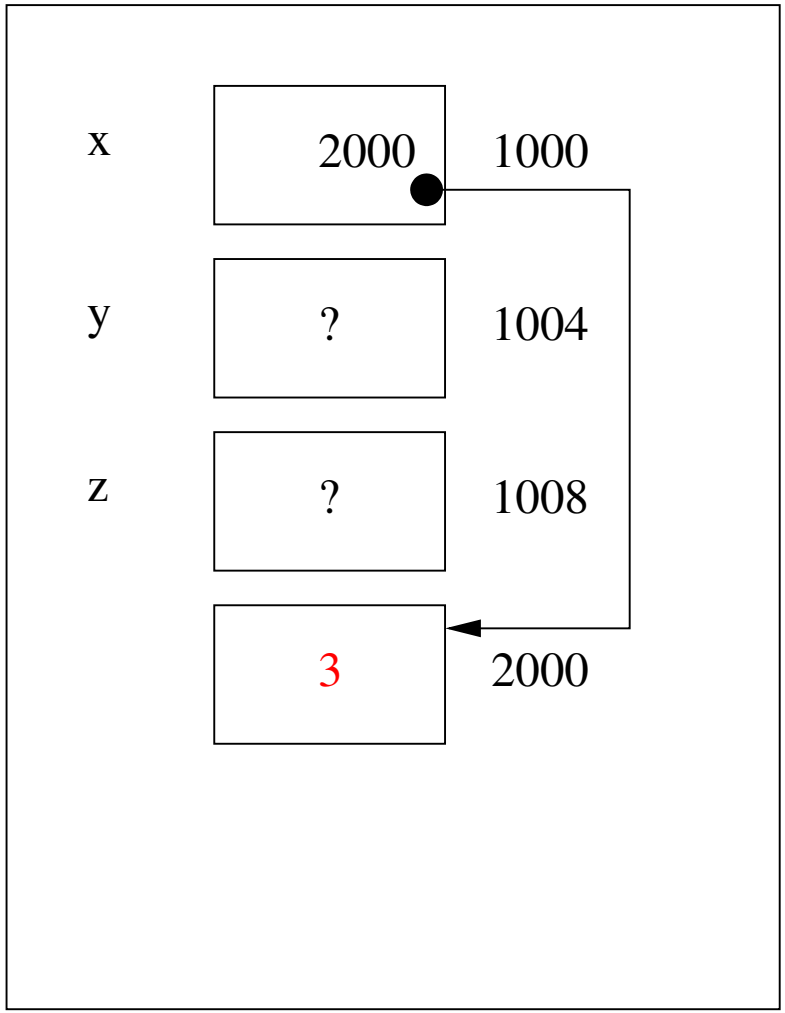
C++ Pointer Example 2

```
int *x, *y, *z;
x = new int;
*x = 3;
y = new int;
*y = 4;
z = x;
x = y;
delete z;
delete y;
```



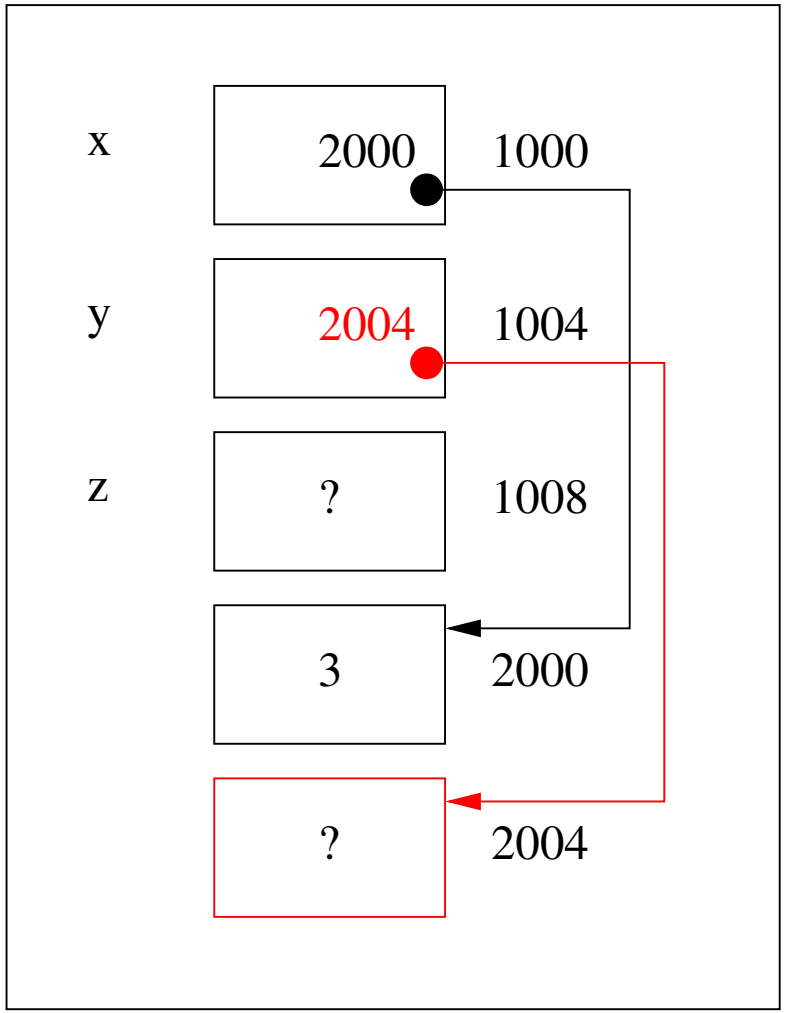
C++ Pointer Example 2

```
int *x, *y, *z;
x = new int;
*x = 3;
y = new int;
*y = 4;
z = x;
x = y;
delete z;
delete y;
```



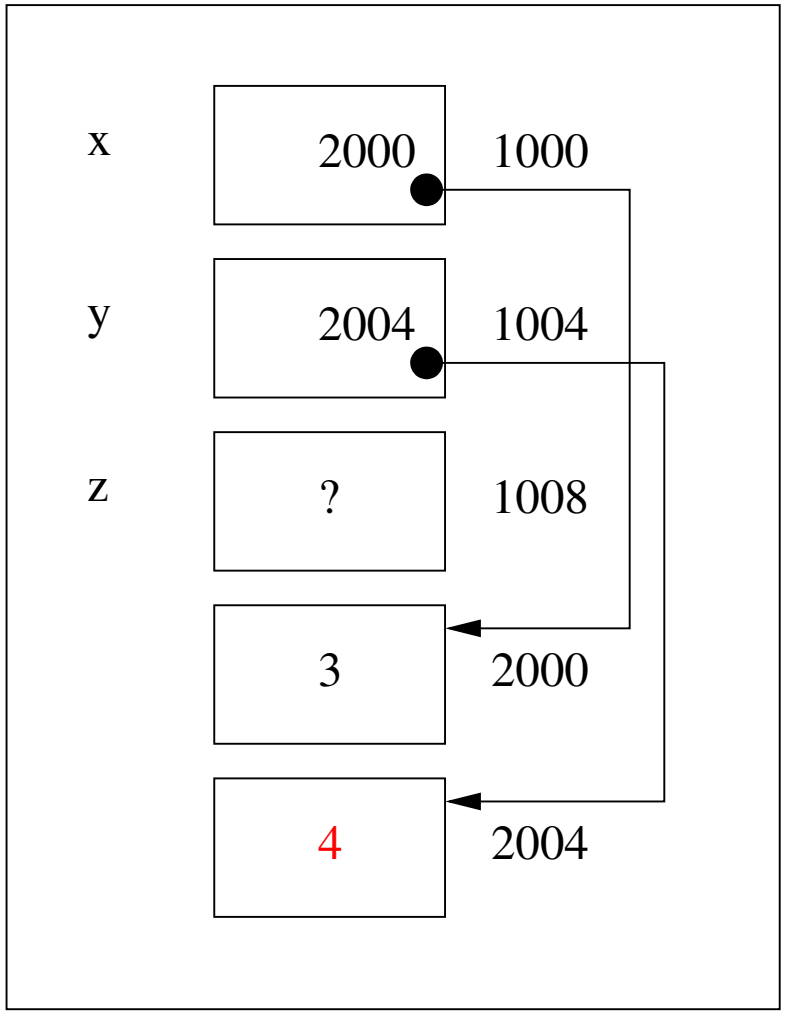
C++ Pointer Example 2

```
int *x, *y, *z;
x = new int;
*x = 3;
y = new int;
*y = 4;
z = x;
x = y;
delete z;
delete y;
```



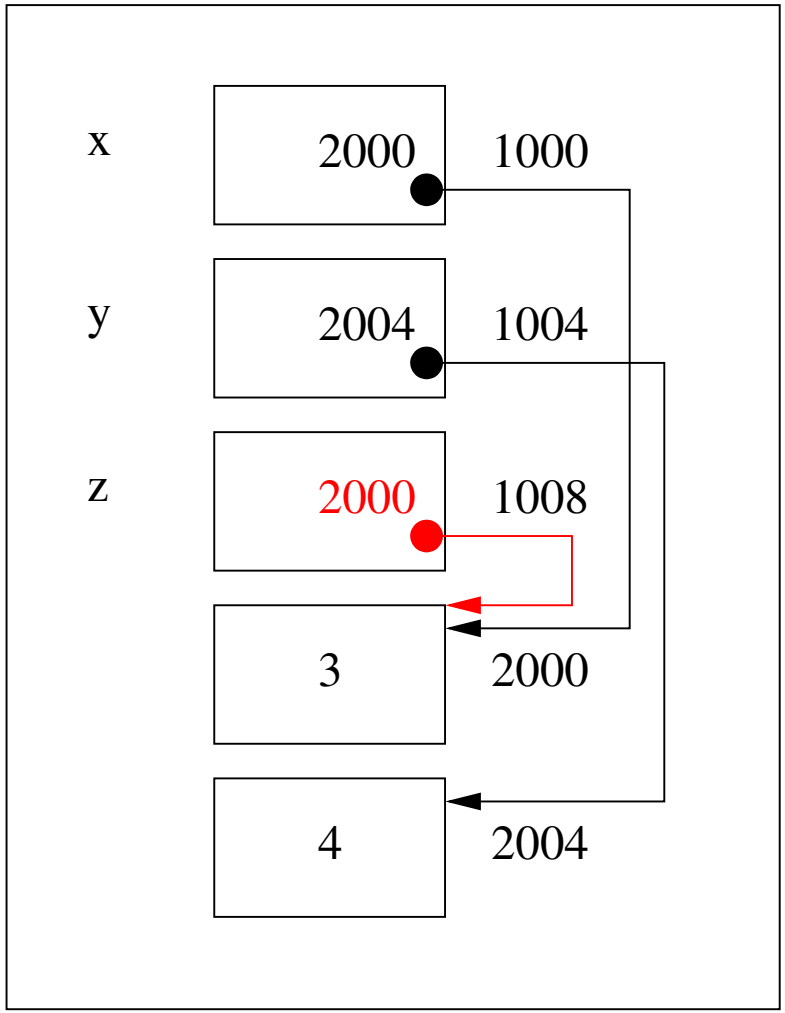
C++ Pointer Example 2

```
int *x, *y, *z;
x = new int;
*x = 3;
y = new int;
*y = 4;
z = x;
x = y;
delete z;
delete y;
```



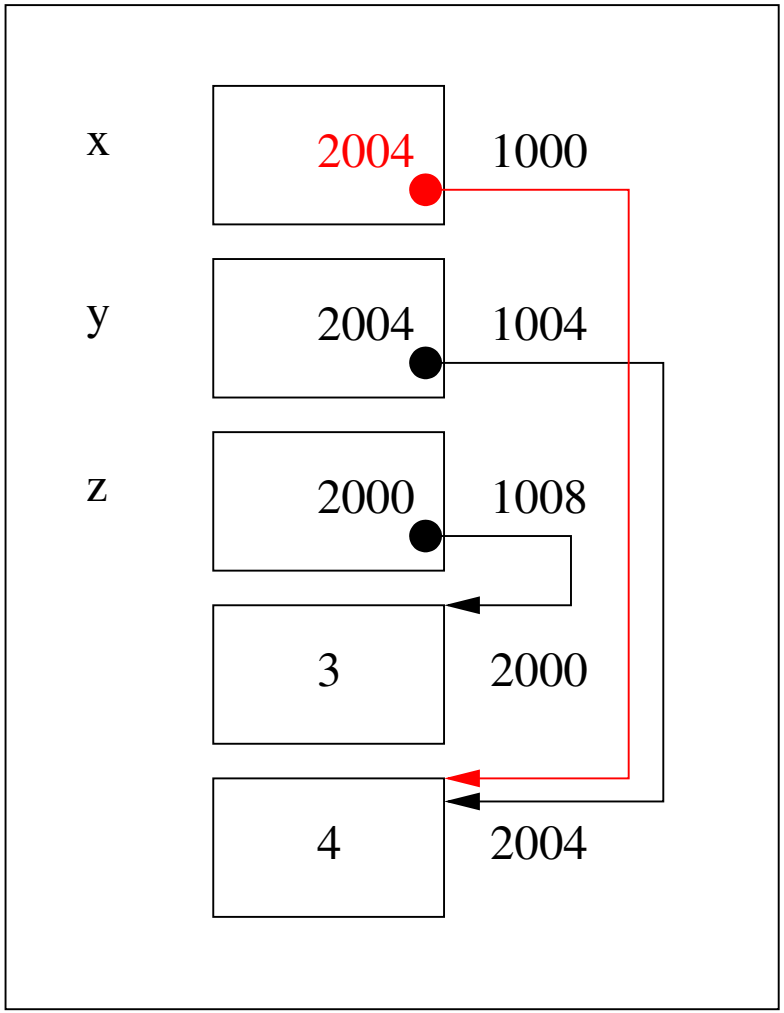
C++ Pointer Example 2

```
int *x, *y, *z;
x = new int;
*x = 3;
y = new int;
*y = 4;
z = x;
x = y;
delete z;
delete y;
```



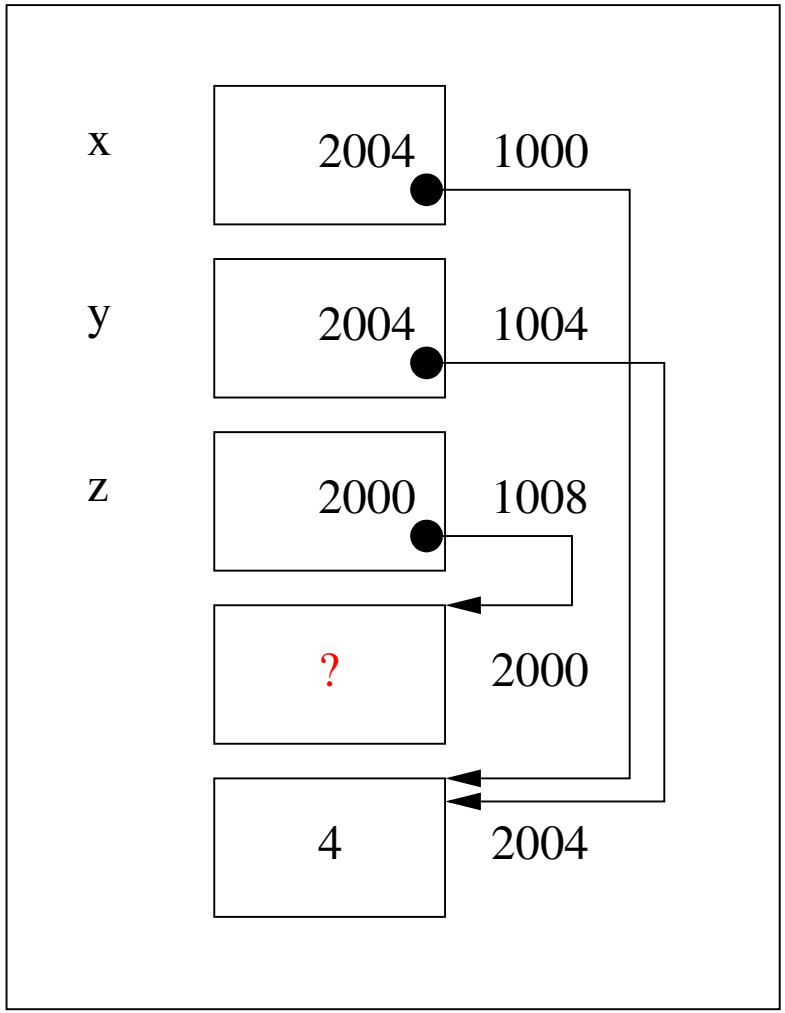
C++ Pointer Example 2

```
int *x, *y, *z;
x = new int;
*x = 3;
y = new int;
*y = 4;
z = x;
x = y;
delete z;
delete y;
```



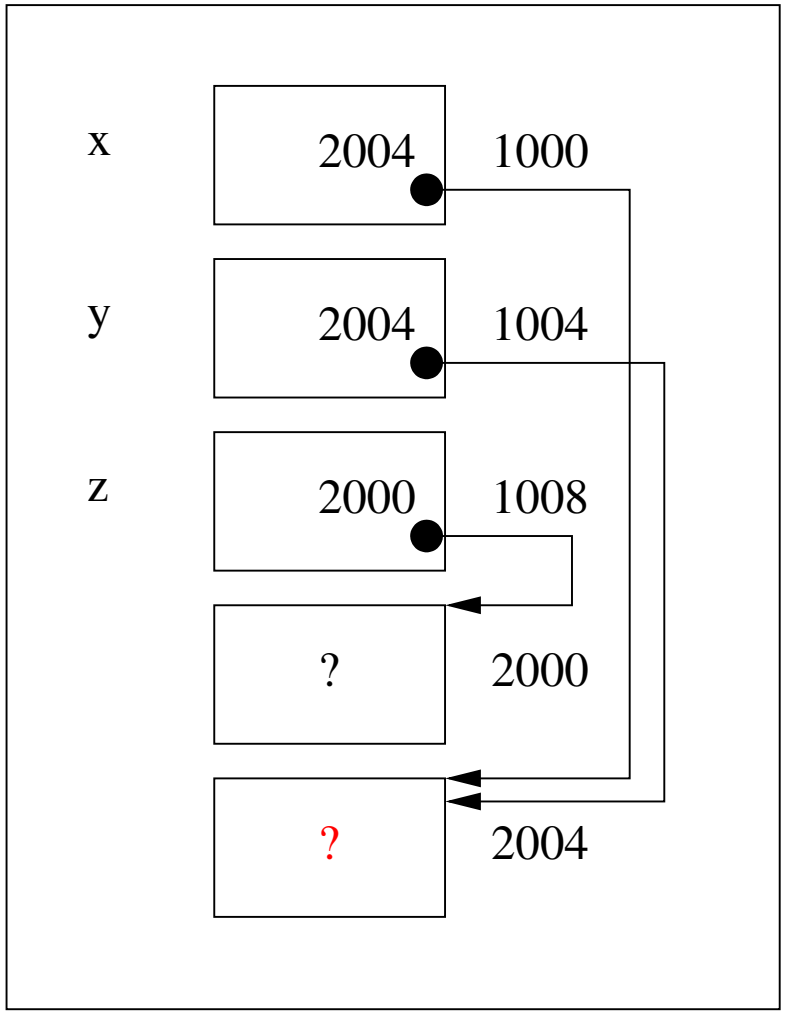
C++ Pointer Example 2

```
int *x, *y, *z;
x = new int;
*x = 3;
y = new int;
*y = 4;
z = x;
x = y;
delete z;
delete y;
```



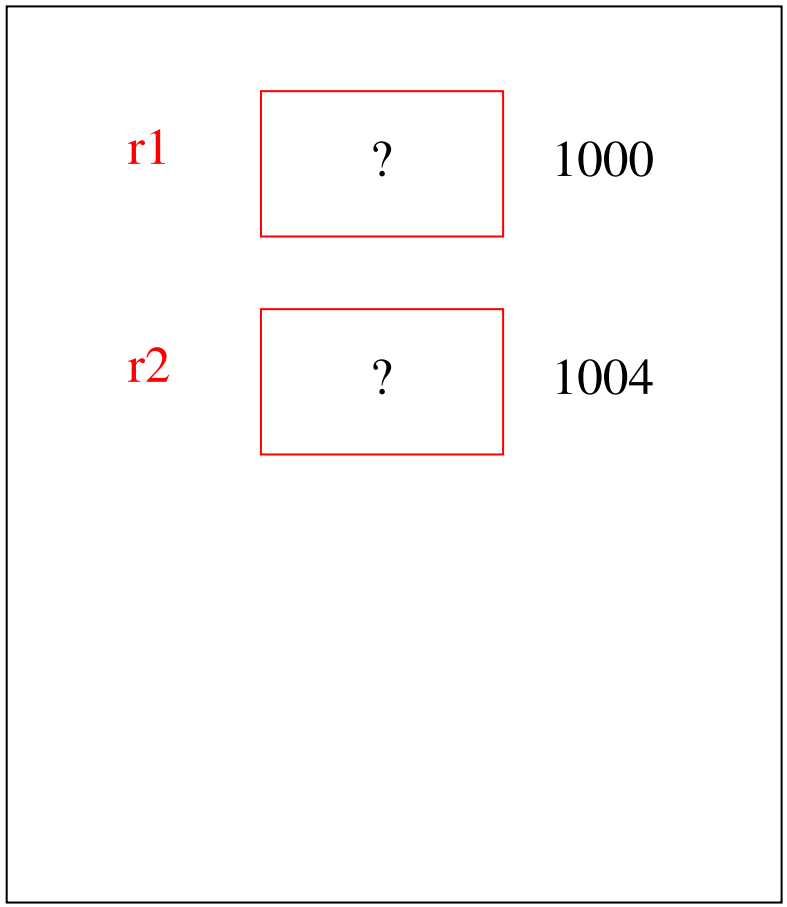
C++ Pointer Example 2

```
int *x, *y, *z;
x = new int;
*x = 3;
y = new int;
*y = 4;
z = x;
x = y;
delete z;
delete y;
```



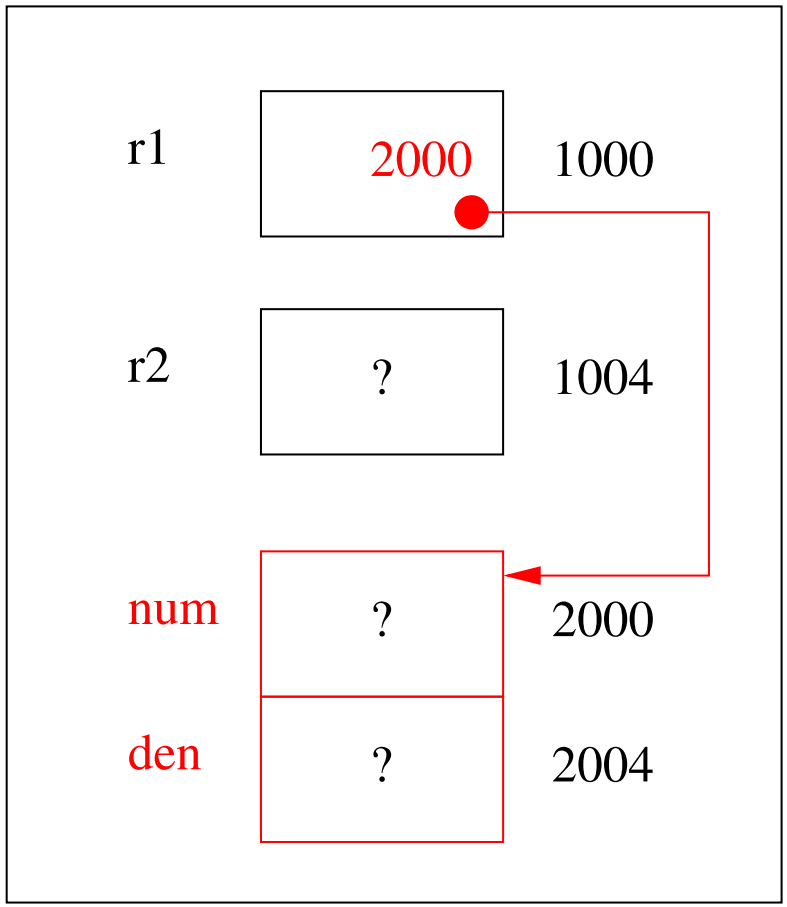
C++ Pointer Example 3

```
Rational *r1, *r2;
r1 = new Rational;
r1->set(2, 3);
r2 = r1;
r1->set(1, 3);
delete r1;
```



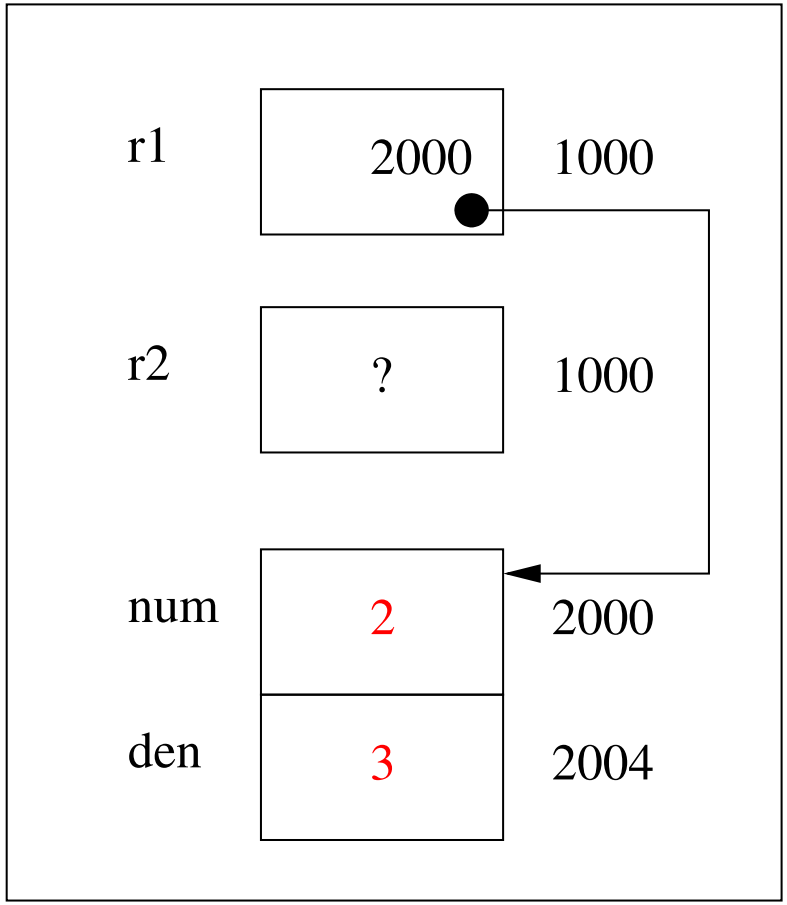
C++ Pointer Example 3

```
Rational *r1, *r2;
r1 = new Rational;
r1->set(2, 3);
r2 = r1;
r1->set(1, 3);
delete r1;
```



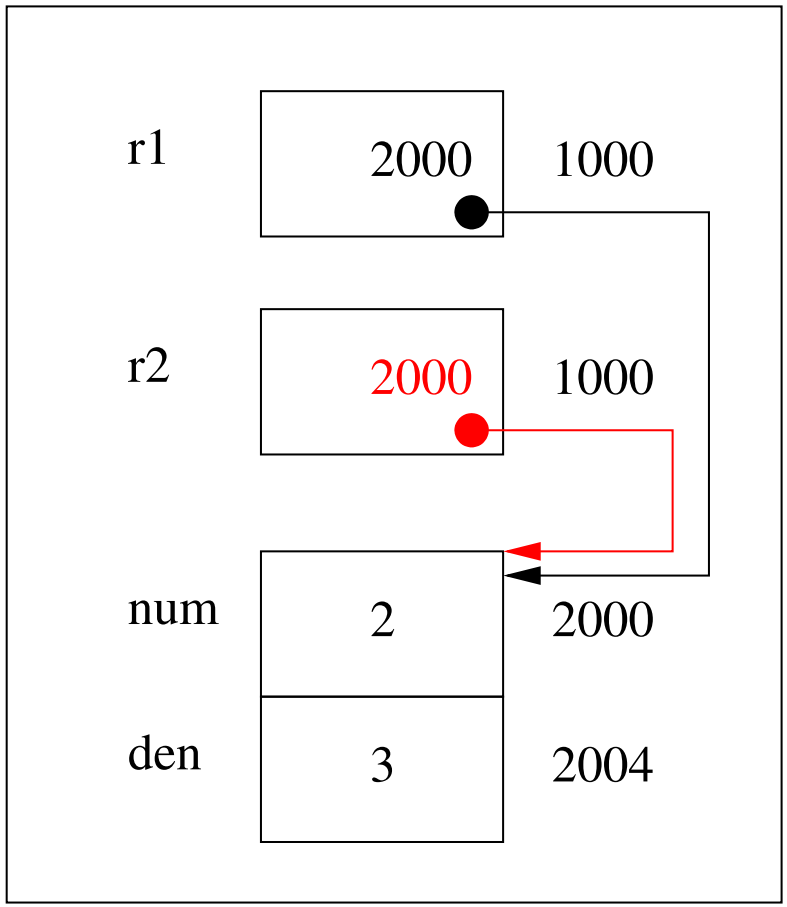
C++ Pointer Example 3

```
Rational *r1, *r2;
r1 = new Rational;
r1->set(2, 3);
r2 = r1;
r1->set(1, 3);
delete r1;
```



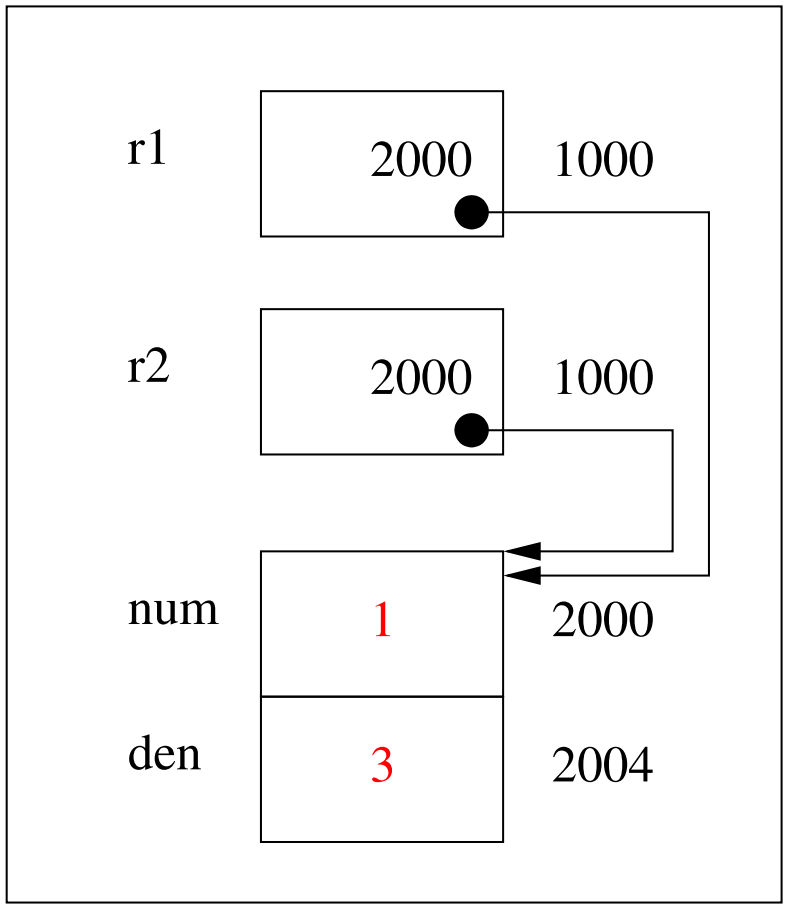
C++ Pointer Example 3

```
Rational *r1, *r2;
r1 = new Rational;
r1->set(2, 3);
r2 = r1;
r1->set(1, 3);
delete r1;
```



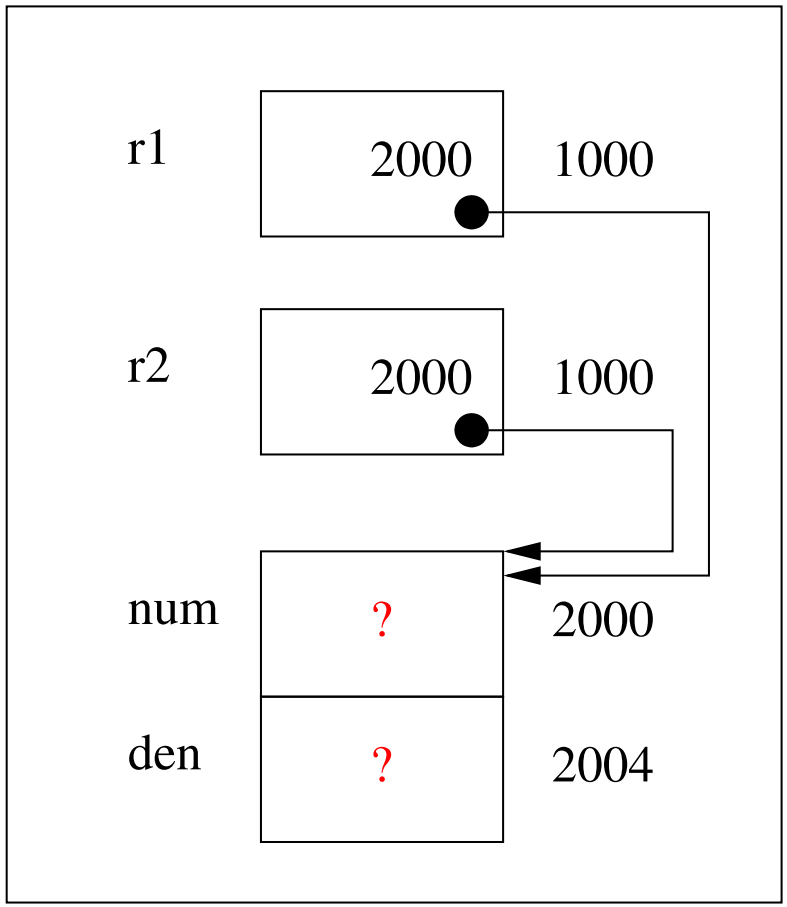
C++ Pointer Example 3

```
Rational *r1, *r2;
r1 = new Rational;
r1->set(2, 3);
r2 = r1;
r1->set(1, 3);
delete r1;
```



C++ Pointer Example 3

```
Rational *r1, *r2;
r1 = new Rational;
r1->set(2, 3);
r2 = r1;
r1->set(1, 3);
delete r1;
```



C++ Arrays

Static Arrays

- Internally, a is a **pointer** to the first item in the array, `a[0]`.
- The size is fixed at compile time. Here, it is 10: (`int a[10];`
- A static array cannot be expanded to a larger capacity.

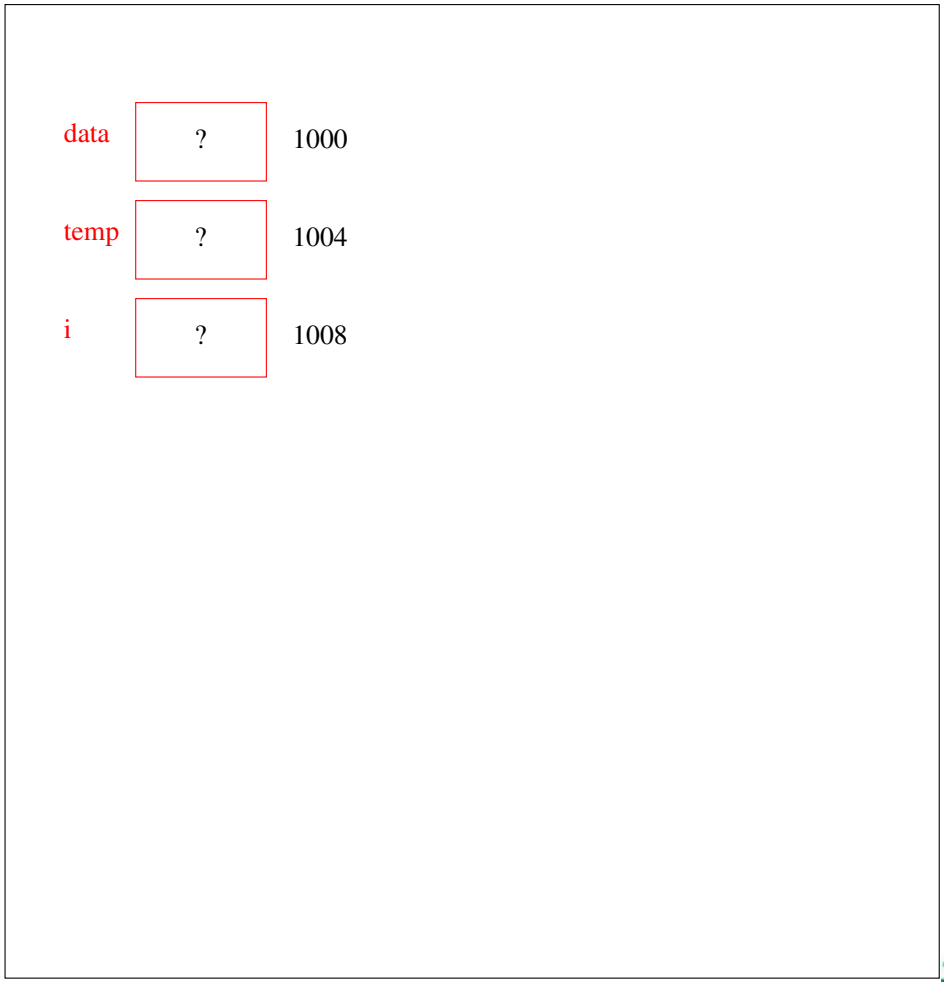
C++ Arrays

Dynamic Arrays

- A dynamic array is explicitly declared as a pointer:
  `int *a;`
- It is given an initial size using the `new` operator:
  `a = new int[5];`
- A dynamic array can be expanded: Its items can be copied into a larger area, whose address can be assigned to the original variable.

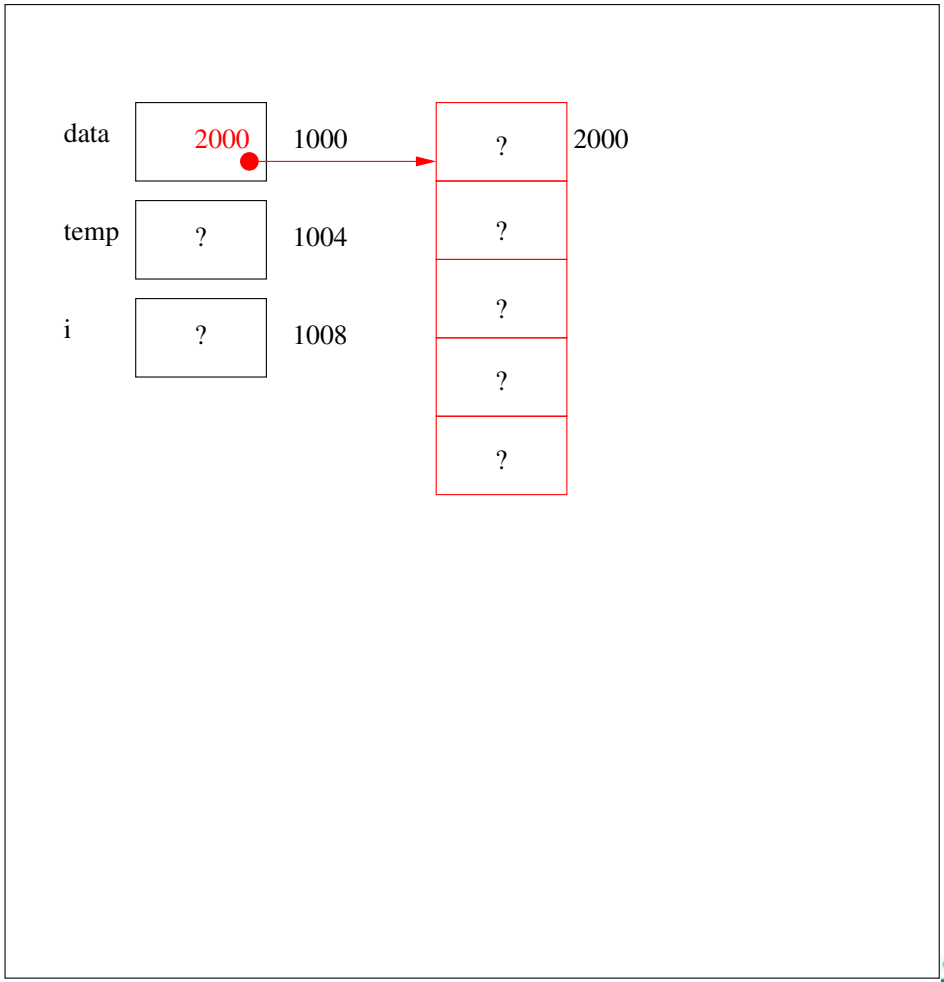
C++ Dynamic Array Example

```
int *data, *temp, i;
data = new int[5];
for (i=0; i<5; ++i) {
   data[i] = i;
}
temp = new int[10];
for (i=0; i<5; ++i) {
   temp[i] = data[i];
}
delete [] data;
data = temp;
for (i=5; i<10; ++i) {
   data[i] = i;
}
delete [] data;
```



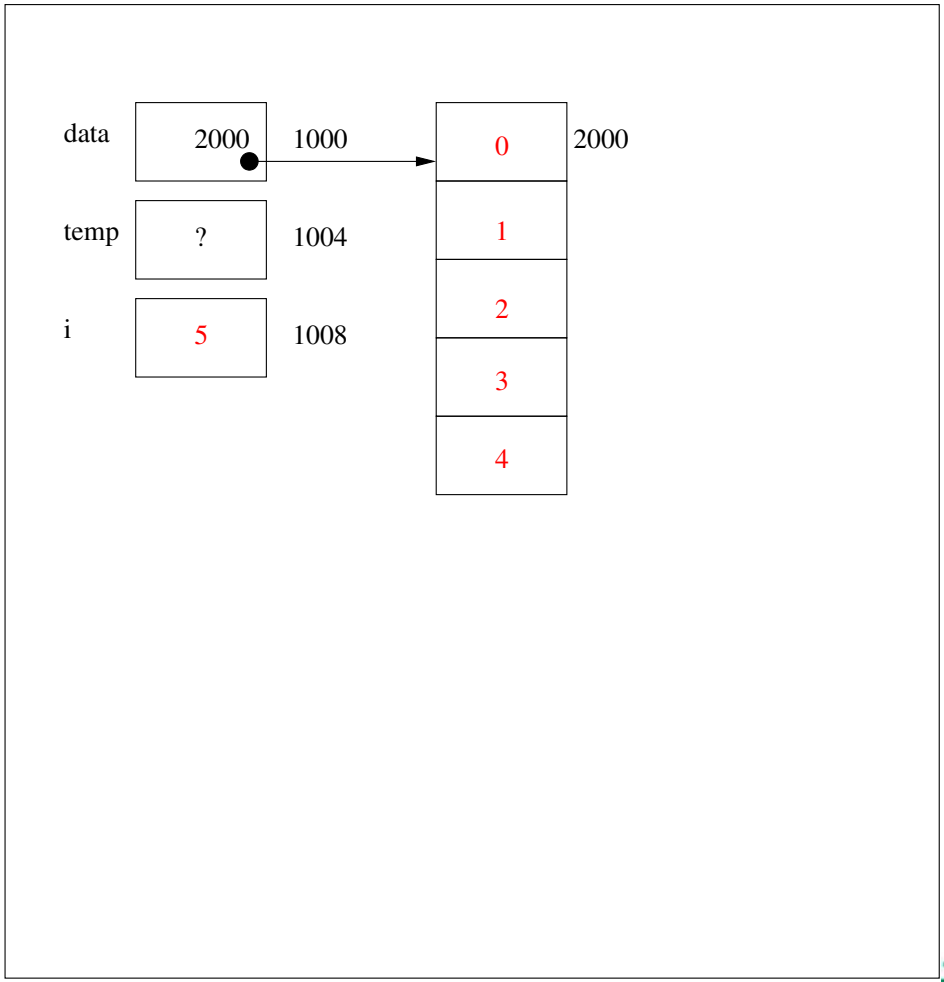
C++ Dynamic Array Example

```
int *data, *temp, i;
data = new int[5];
for (i=0; i<5; ++i) {
   data[i] = i;
}
temp = new int[10];
for (i=0; i<5; ++i) {
   temp[i] = data[i];
}
delete [] data;
data = temp;
for (i=5; i<10; ++i) {
   data[i] = i;
}
delete [] data;
```



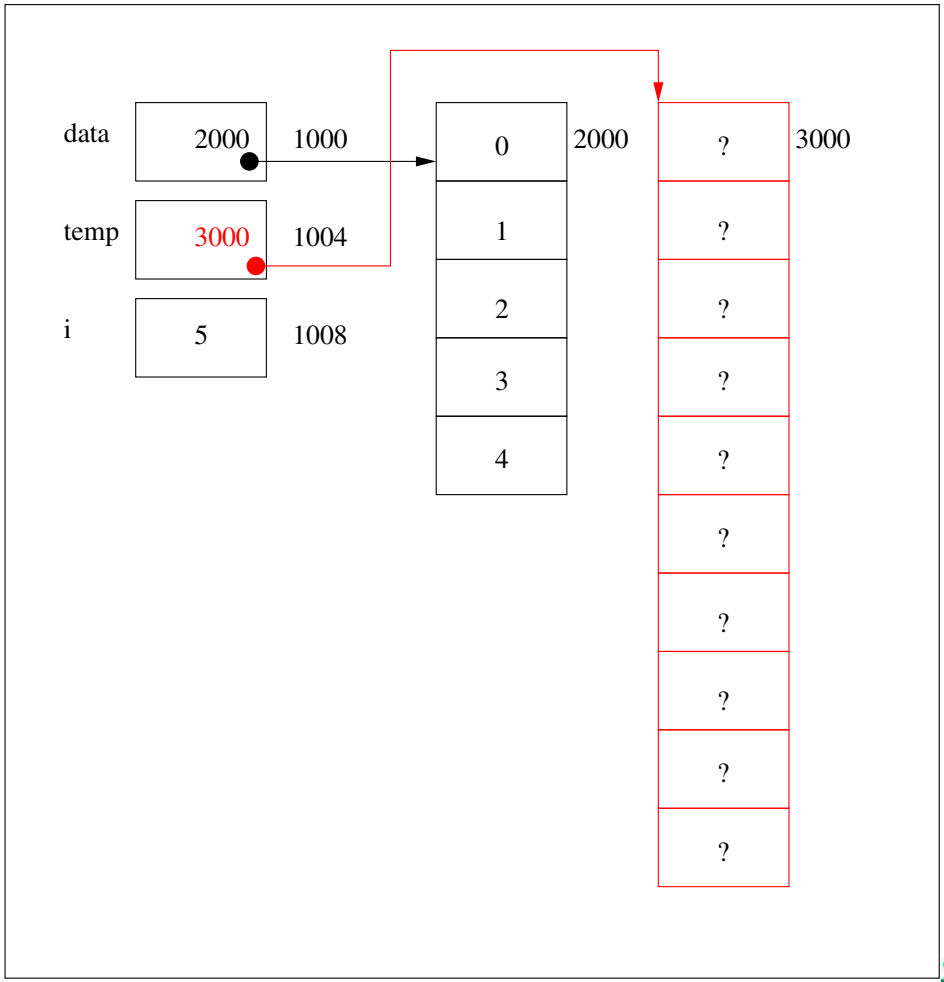
C++ Dynamic Array Example

```
int *data, *temp, i;
data = new int[5];
for (i=0; i<5; ++i) {
   data[i] = i;
}
temp = new int[10];
for (i=0; i<5; ++i) {
   temp[i] = data[i];
}
delete [] data;
data = temp;
for (i=5; i<10; ++i) {
   data[i] = i;
}
delete [] data;
```



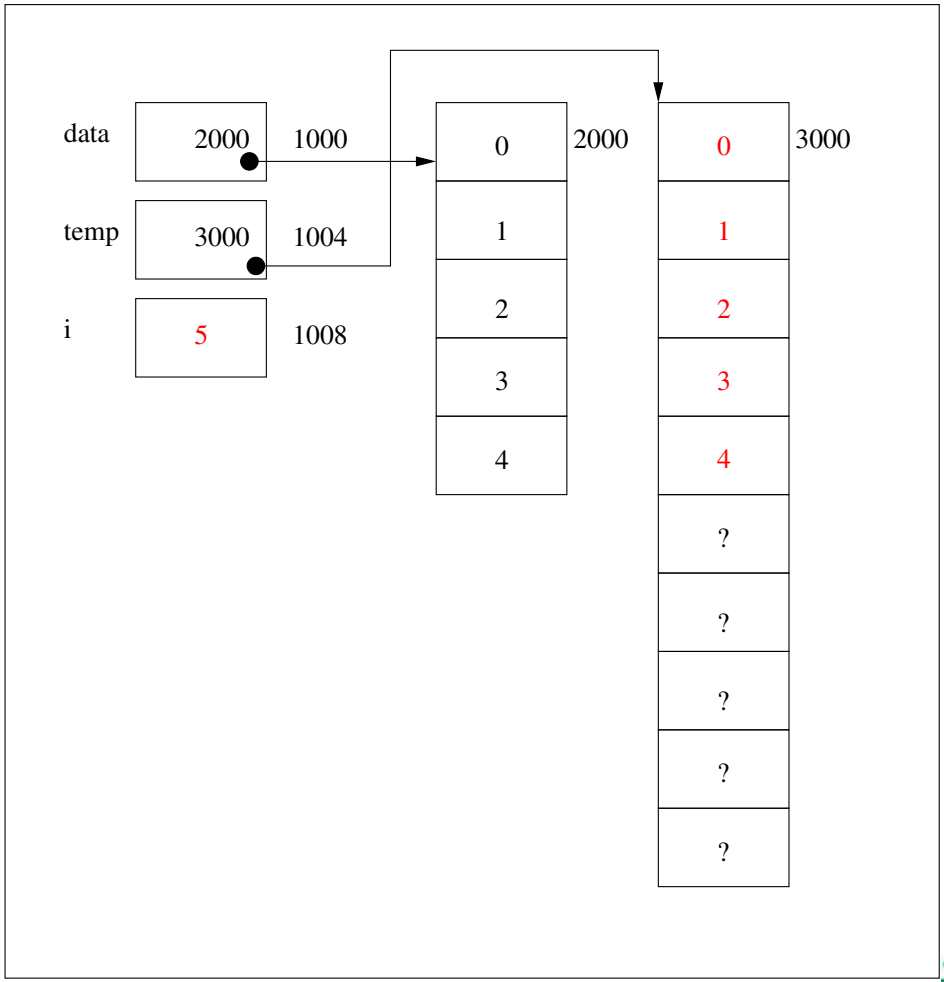
C++ Dynamic Array Example

```
int *data, *temp, i;
data = new int[5];
for (i=0; i<5; ++i) {
   data[i] = i;
}
temp = new int[10];
for (i=0; i<5; ++i) {
   temp[i] = data[i];
}
delete [] data;
data = temp;
for (i=5; i<10; ++i) {
   data[i] = i;
}
delete [] data;
```



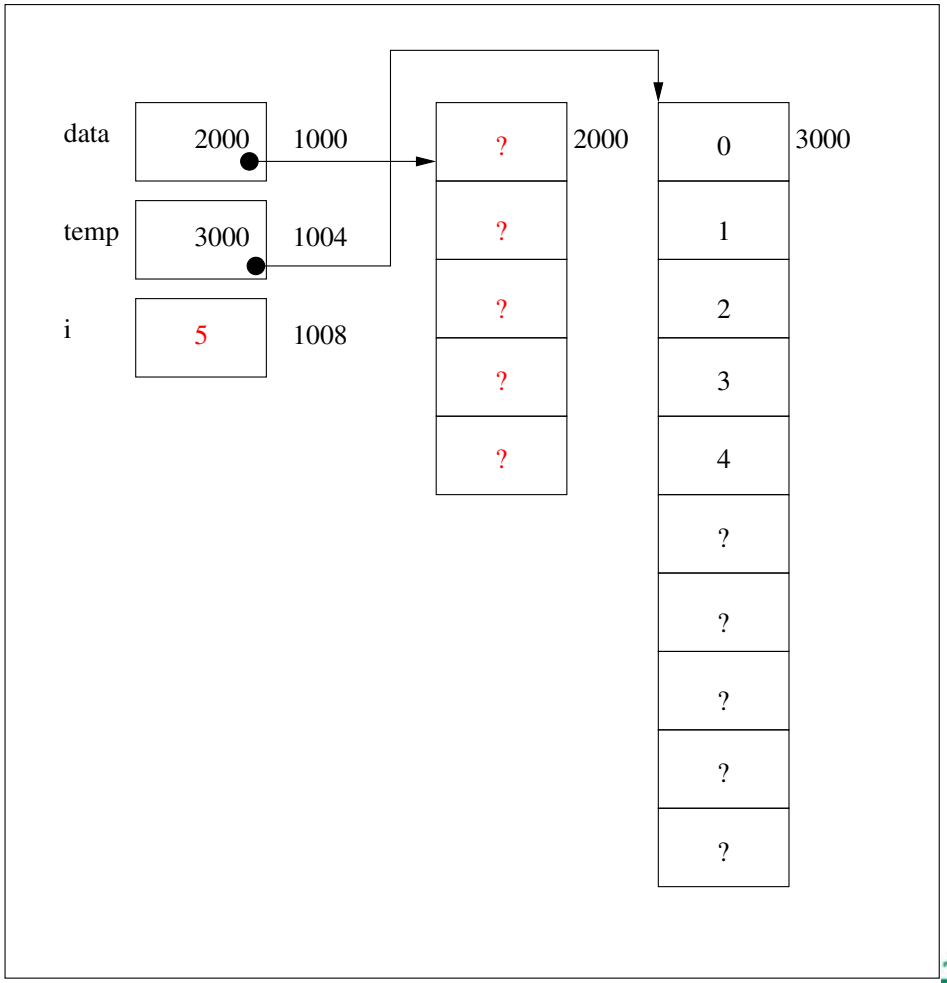
C++ Dynamic Array Example

```
int *data, *temp, i;
data = new int[5];
for (i=0; i<5; ++i) {
   data[i] = i;
}
temp = new int[10];
for (i=0; i<5; ++i) {
   temp[i] = data[i];
}
delete [] data;
data = temp;
for (i=5; i<10; ++i) {
   data[i] = i;
}
delete [] data;
```



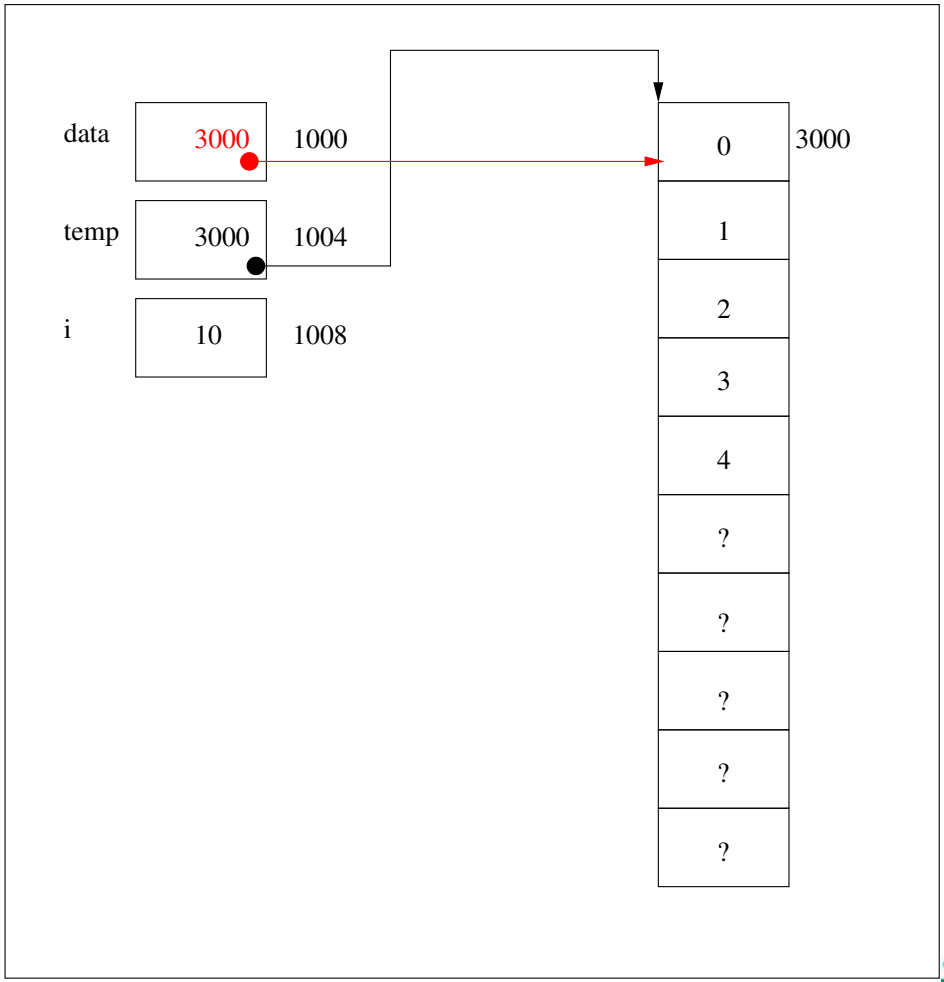
C++ Dynamic Array Example

```
int *data, *temp, i;
data = new int[5];
for (i=0; i<5; ++i) {
   data[i] = i;
}
temp = new int[10];
for (i=0; i<5; ++i) {
   temp[i] = data[i];
}
delete [] data;
data = temp;
for (i=5; i<10; ++i) {
   data[i] = i;
}
delete [] data;
```



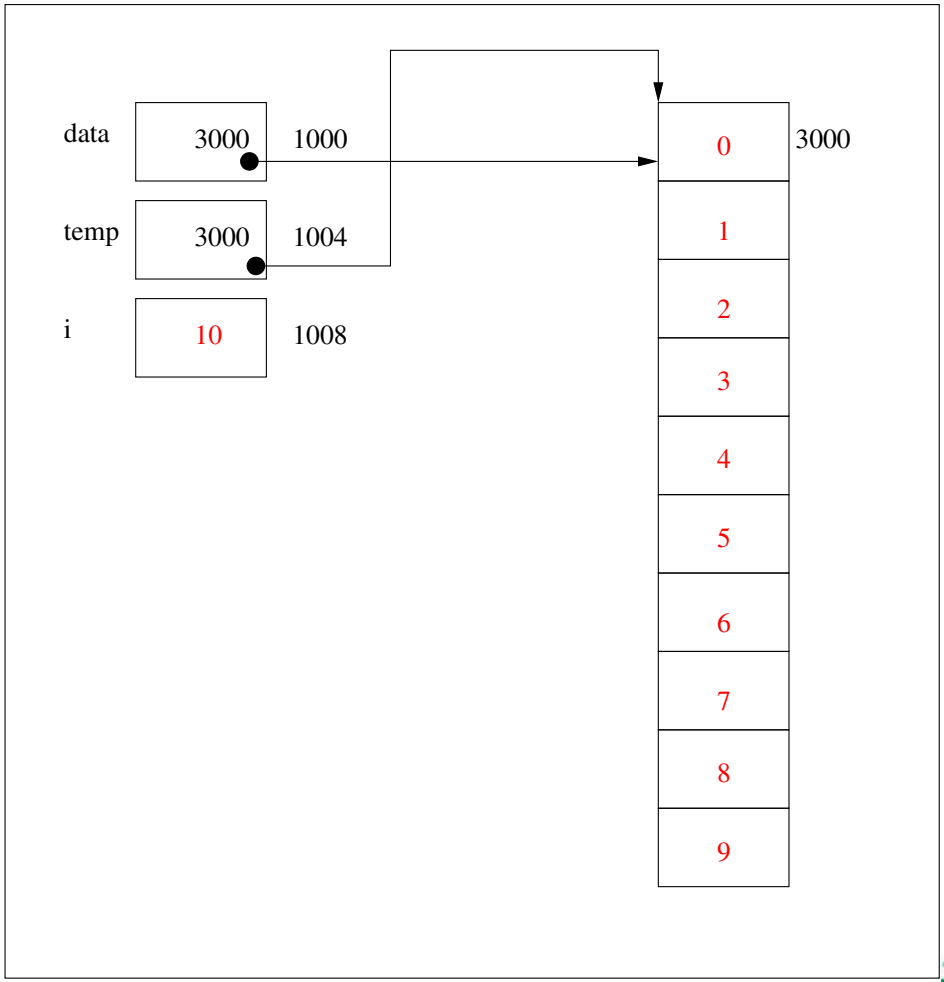
C++ Dynamic Array Example

```
int *data, *temp, i;
data = new int[5];
for (i=0; i<5; ++i) {
   data[i] = i;
}
temp = new int[10];
for (i=0; i<5; ++i) {
   temp[i] = data[i];
}
delete [] data;
data = temp;
for (i=5; i<10; ++i) {
   data[i] = i;
}
delete [] data;
```



C++ Dynamic Array Example

```
int *data, *temp, i;
data = new int[5];
for (i=0; i<5; ++i) {
   data[i] = i;
}
temp = new int[10];
for (i=0; i<5; ++i) {
   temp[i] = data[i];
}
delete [] data;
data = temp;
for (i=5; i<10; ++i) {
   data[i] = i;
}
delete [] data;
```



C++ Dynamic Array Example

```
int *data, *temp, i;
data = new int[5];
for (i=0; i<5; ++i) {
   data[i] = i;
}
temp = new int[10];
for (i=0; i<5; ++i) {
   temp[i] = data[i];
}
delete [] data;
data = temp;
for (i=5; i<10; ++i) {
   data[i] = i;
}
delete [] data;
```