

Chapter-3

Pointers

Pointer

- A pointer is a variable that holds the *address* of something else.
- The only difference between pointer variable and regular variable is the data they hold.
- There are two pointer operators in C++:
 - & the address of operator
 - * the dereference operator
- We are going to suppose that **var** has been placed in the memory address **1776** and that we write the following:
- `var=25; x=var;`
- `ptr = &var;`

- **Syntax** :

dataType *identifier;

e.g.

int *x;

x is a pointer to an integer.

- Addresses: garbage, something, nothing

- When created:

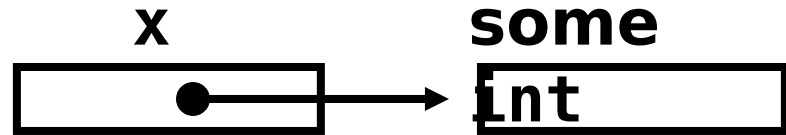
int * q;

q = 0; // now it points to nothing

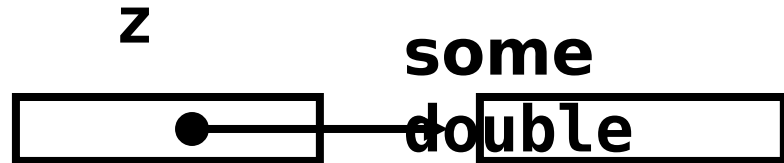
q = NULL; // not portable, use 0 instead

Pointers to anything

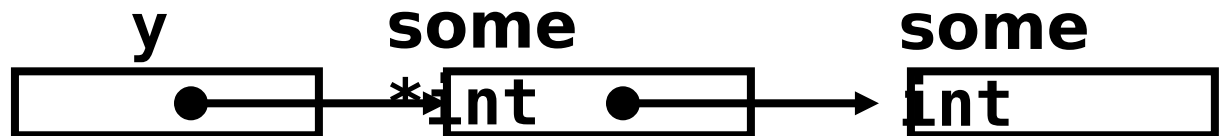
int *x;



double *z;



int **y;



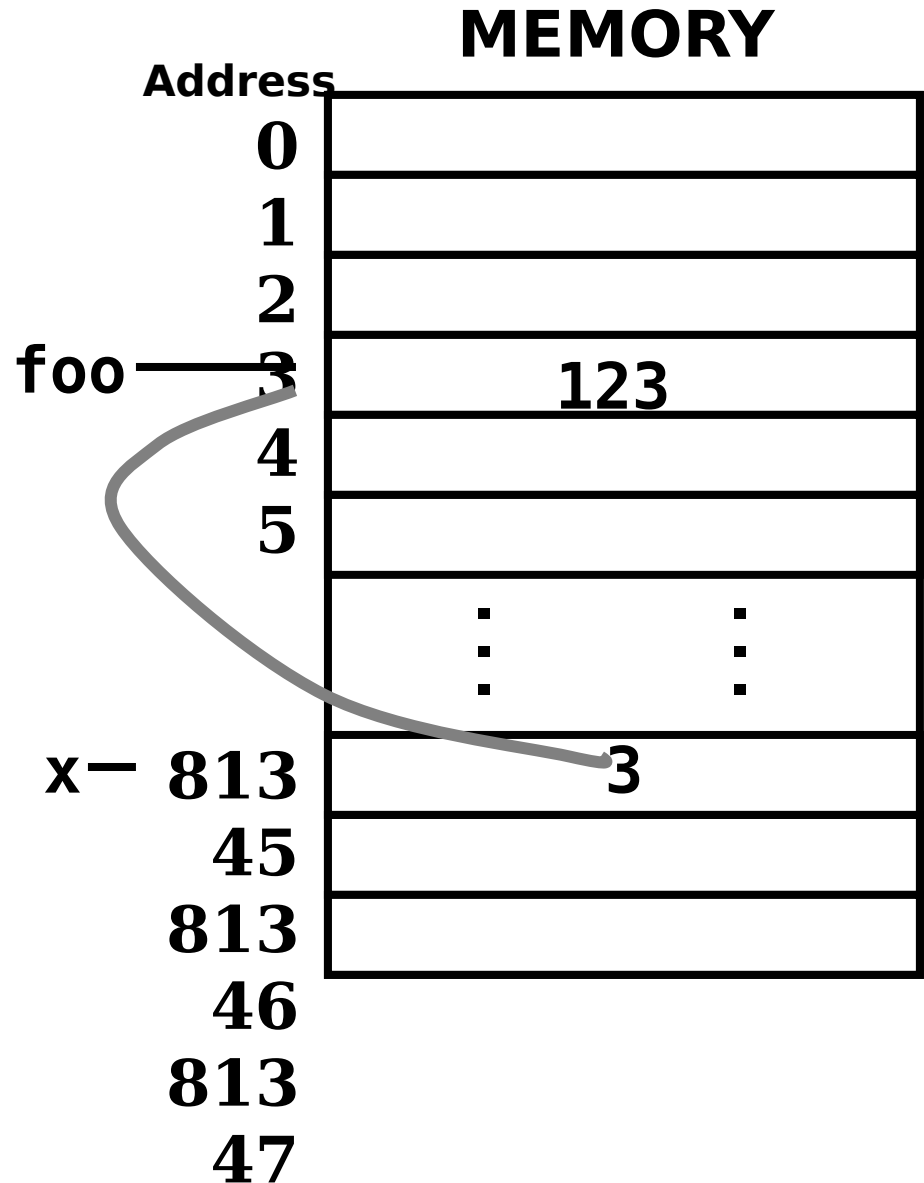
- To assign a value to the pointer variable
 - use the address operator &
- E.g.

```
int foo;
```

```
int *x;
```

```
foo = 123;
```

```
x = &foo;
```



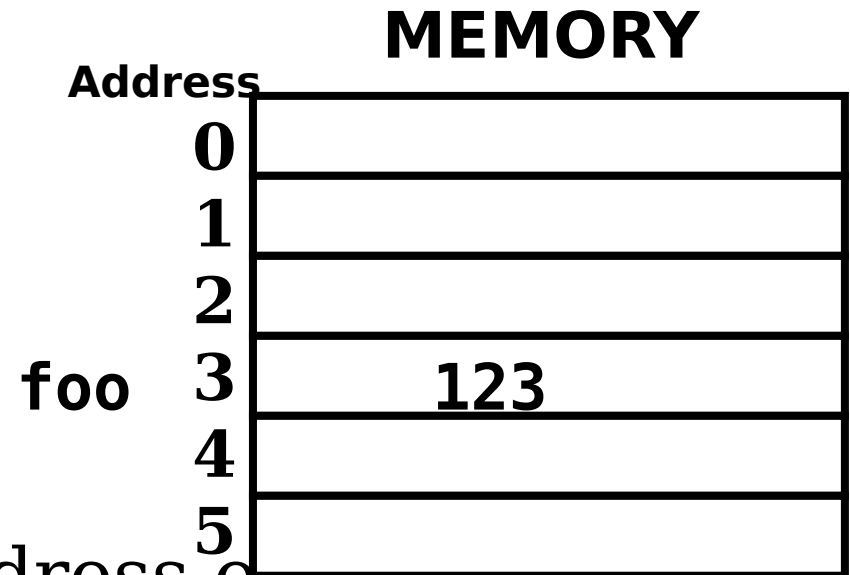
&foo

In C++ you can get the *address* of a variable with the “&” operator.

```
int foo;
```

```
foo = 123;
```

```
x = &foo;
```



- **&foo** means “the address of foo”
- Must always refer to something
 - Must be initialized, cannot be changed

Dereferencing operator

- To assign a value to the variable foo through the pointer variable
 - we must use the dereferencing operator *
 - E.g

```
*x=10;
```

//Changes the value of foo from 123 to 10

Equivalent to

```
foo=10;
```

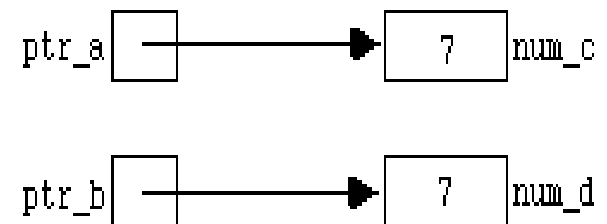
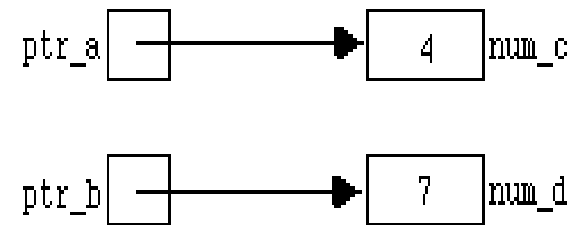
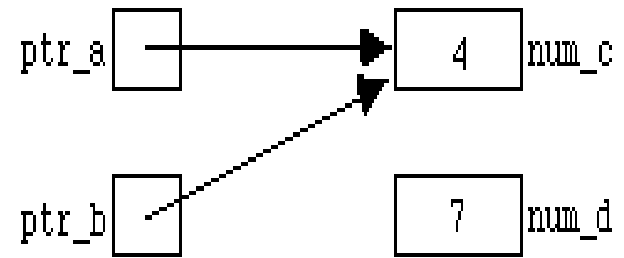
- You can use the integer x points to in a C++ expression like this:

```
y = *x + 17;
```

```
*x = *x + 1;
```

```
#include <iostream.h>
```

```
int main() {  
    int *ptr_a, *ptr_b;  
    int num_c = 4, num_d = 7;  
    ptr_a = &num_c;  
    ptr_b = ptr_a;  
    cout << *ptr_a << " " << *ptr_b  
        << "\n";  
    ptr_b = &num_d;  
    cout << *ptr_a << " " << *ptr_b  
        << "\n";  
    *ptr_a = *ptr_b;  
    cout << *ptr_a << " " << *ptr_b  
        << "\n";  
    cout << num_c << " " <<  
        *&*&*&num_c << "\n";  
    return 0; }
```



• The output of this program is:

- 4 4
- 4 7
- 7 7

More Pointers

```
int i = 50;
```

•

```
int j = 75;
```

```
int *p1 ; int * p2
```

```
p1 = &i ; p2 = &j;
```

```
cout << *p1;
```

```
p1 = p2 ; *p2 =0;
```

```
cout <<*p1;
```

Assigning a value to a *dereferenced* pointer

A pointer must have a value before you can *dereference* it (follow the pointer).

```
int  
*x;  
*x=3
```

ERROR!!!
x doesn't point to
anything!!!



```
int foo;  
int *x;  
x = &foo;  
*x=3;
```

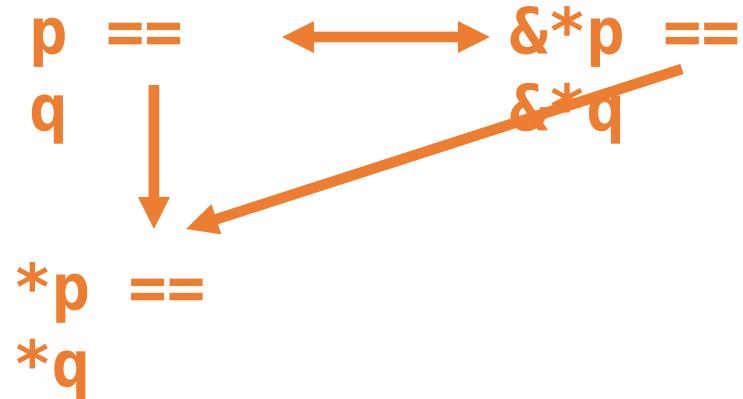
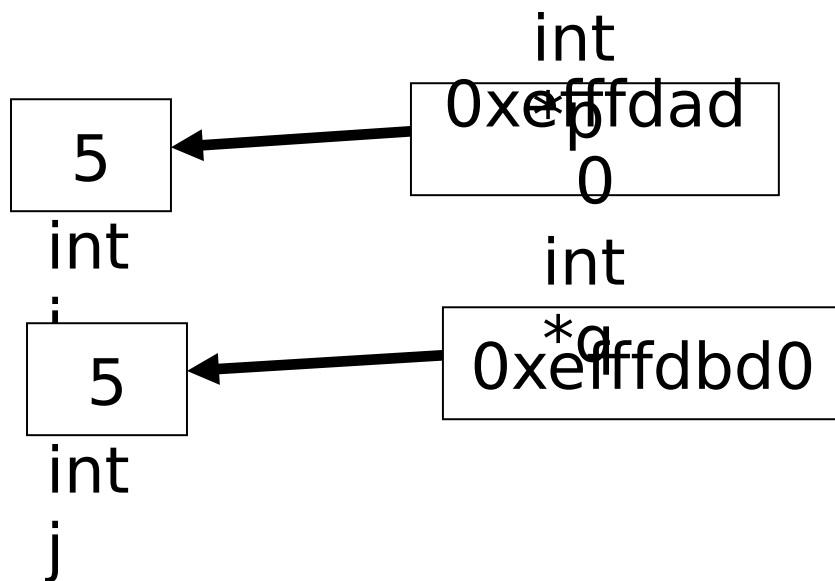
this is fine
x points to
foo



Location and Value

Comparisons

- Pointers may be compared for equality
 - Same as comparing addresses of what pointers point to (memory locations:)
 - Contents of what pointers point to may be compared, too
- First implies second, not other way around



Const Pointers and Pointers to Const Types

```
int main ( ){  
    const int i = 0;  
    int j = 1;  
    int k = 2;  
  
    // pointer to int  
    int * w = &j;  
  
    // const pointer to int  
    int * const x = &k;  
  
    // pointer to const int  
    const int * y = &i;  
  
    // const pointer to const int  
    const int * const z = &j;  
}
```

- Make promises via the **const** keyword in pointer declaration:
 - not to change pointer itself
 - not to change value it aliases
 - can also promise neither/both
- Read declarations right to left
- In this example, can change
 - **w** and what it points to
 - what **x** points to but not **x**
 - **y** but not what it points to
 - neither **z** nor what it points to
- A pointer to a non-const type cannot point to a const variable
 - **w** and **x** can't point to **i**

Pointers and arrays

- The name of an array is a pointer to the 0th element of the array (the beginning of the array)

```
int array[5] ;
```

```
    // array is equivalent to & array[0]
```

```
*array = 5;    is like  array[0] = 5;
```

```
int j = *(array+4)    is like  int j = array[4]
```

```
cout << *array;    is like  cout << array[0];
```

Pointers and Arrays

- An array name is basically a *const* pointer.
- You can use the [] operator with a pointer:

```
int *x;  
int a[10];  
x = &a[2];  
for (int i=0; i<3; i++)  
    x[i]++;
```

 **x** is "the address of **a[2]** "

 **x[i]** is the same as **a[i+2]**

Pointers

- Summary
 - `&p`, `p`, and `*p` all have different meanings
 - `&p` means the address of `p`
 - `p` means the content of `p`
 - `*p` means the content pointed to by `p`, that is pointed to by the content of memory location