CS 161
Intro to CS I

More About Functions:

Default Values, Overloading, and References vs. Pointers

OSU Oregon State University

Odds and Ends…

- Assignment #4 posted
- Demo Assignment #3
- Pythontutor.com (good way to visualize code)

Default Args

access.engr.orst.edu - PuTTY

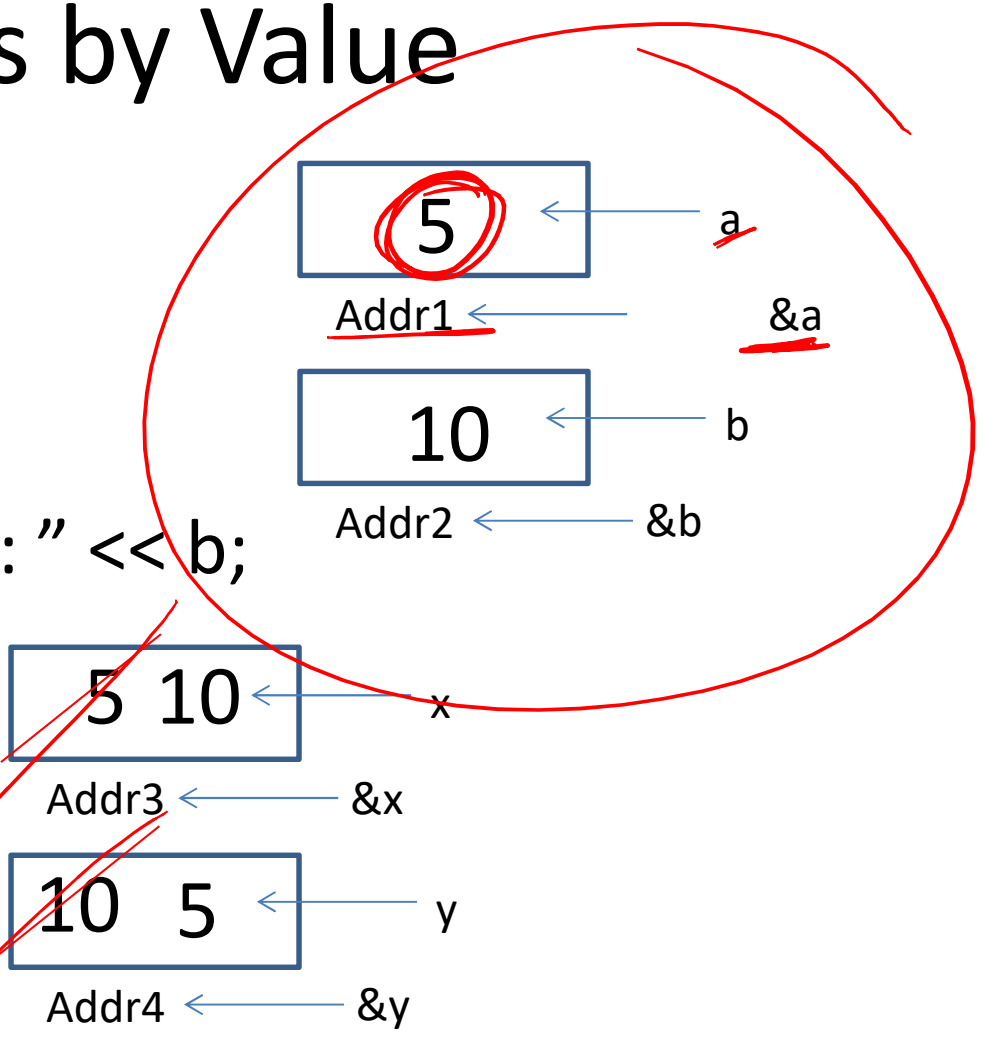
```
 1 #include <iostream>
 2
 3 using std::cout;
 4 using std::endl;
 5
 6 int pwr(int, int n=1);  //Example of default args
 7
 8 int main() {
 9    int base=2, expn=8;
10
11    cout << "The power function: " << pwr(base, expn) << endl;
12    cout << "The power function: " << pwr(base) << endl;
13
14    return 0;
15 }
16
17 int pwr(int x, int n) {
18    int num=1;
19
20    for(int i=0; i < n; i++) {
21       num*=x;
22    }
23
24    return num;
25 }
"test.cpp" 25L, 388C written                  1,19          All
```

C++ Function Overloading

- Multiple functions w/ same name
- Arguments determine function
- Default Args can be done w/ overloading
- Example: pow()
  - http://www.cplusplus.com/reference/cmath/pow/?kw=pow

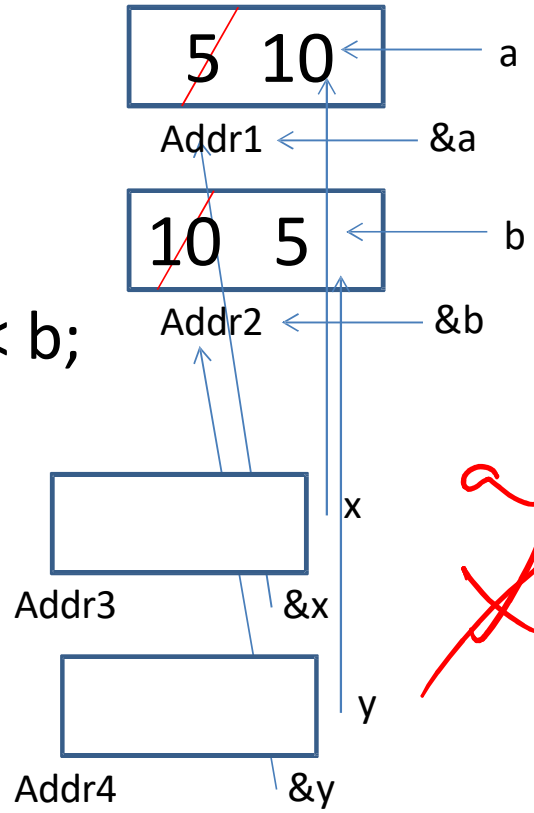
C++ Pass by Value

```
void swap(int, int);
int main() {
   int a=5, b=10;
   swap(a, b);
   cout << "a: " << a << "b: " << b;
}
void swap(int x, int y) {
   int temp = x;
   x = y;
   y = temp;
}
```



C++ Pass by Reference

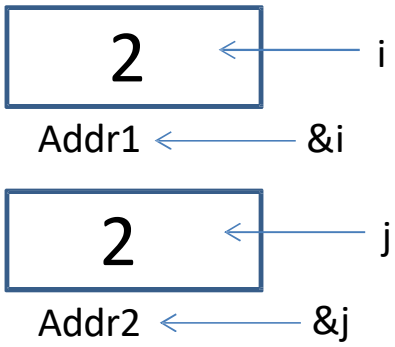
```
void swap(int &, int &);
int main() {
   int a=5, b=10;
   swap(a, b);
   cout << "a: " << a << "b: " << b;
}
void swap(int &x, int &y) {
   int temp = x;
   x = y;
   y = temp;
}
```



Variables vs. Pointers

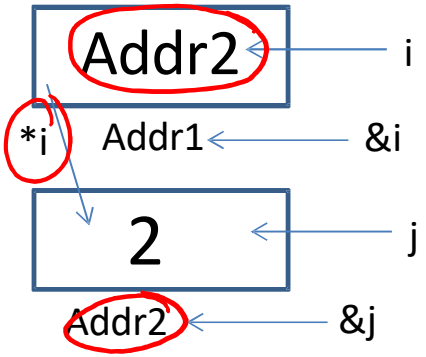
- Value Semantics
  - –Values stored directly
  - –Copy of value is passed

```
int i, j=2;
i=j;
```



- Pointer Semantics
  - – Address to variable is stored
  - – Copy of address is passed

```
int *i, j=2;
i=&j;
```



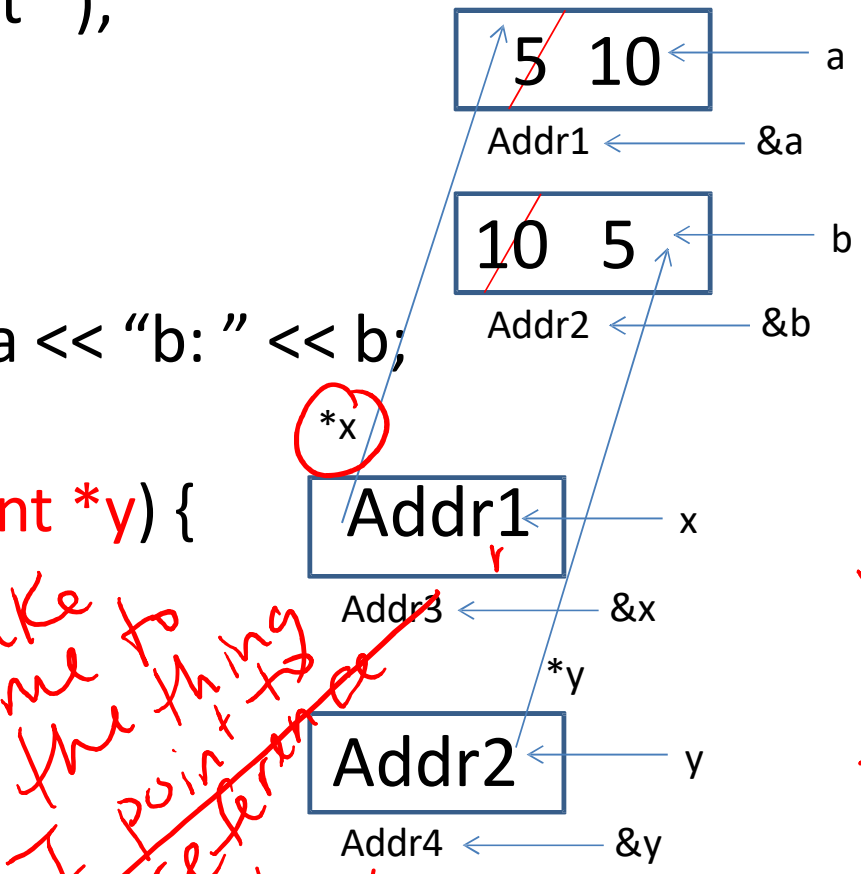
making that i Points to where an int lives

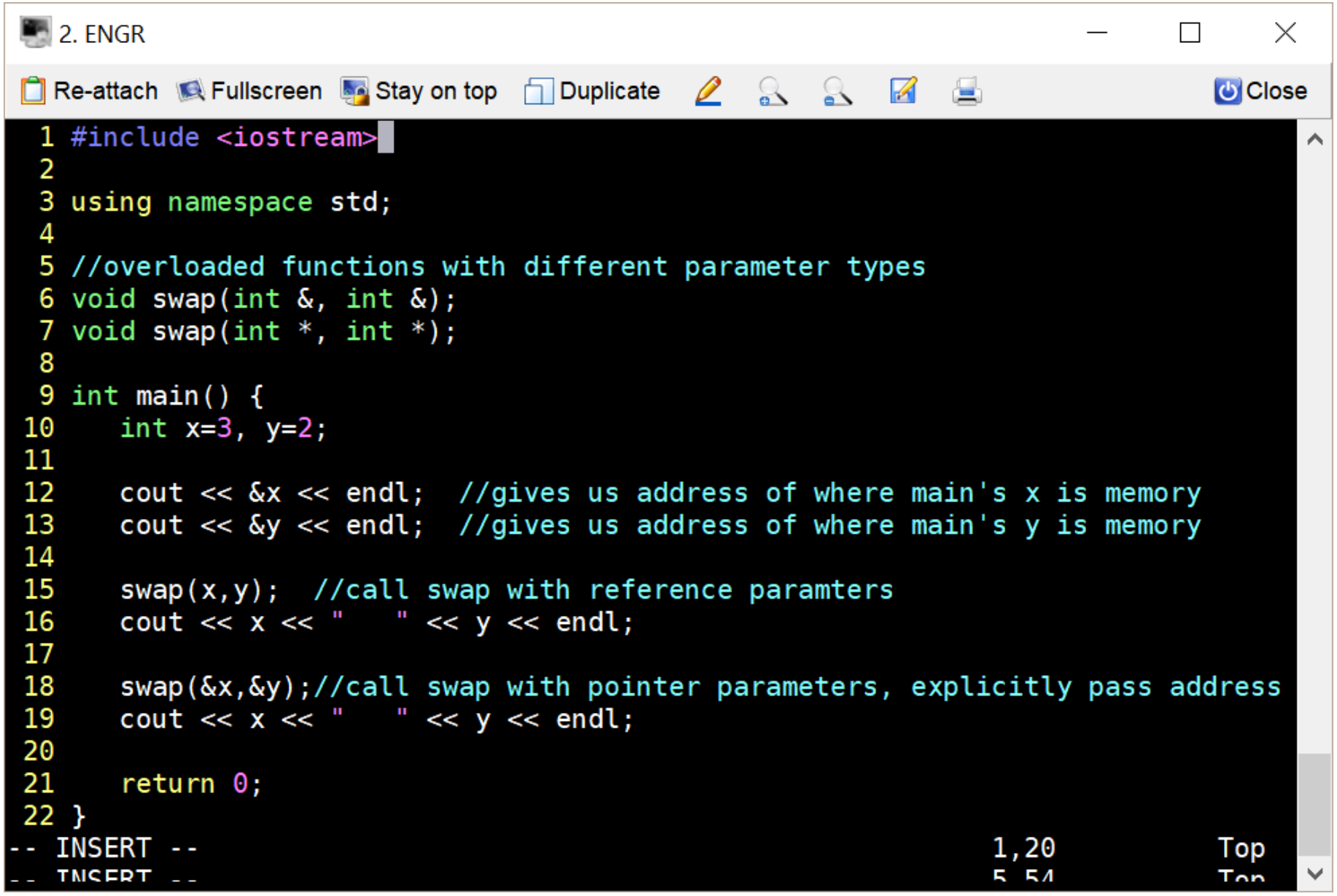
fetch the addr of j & store it in i

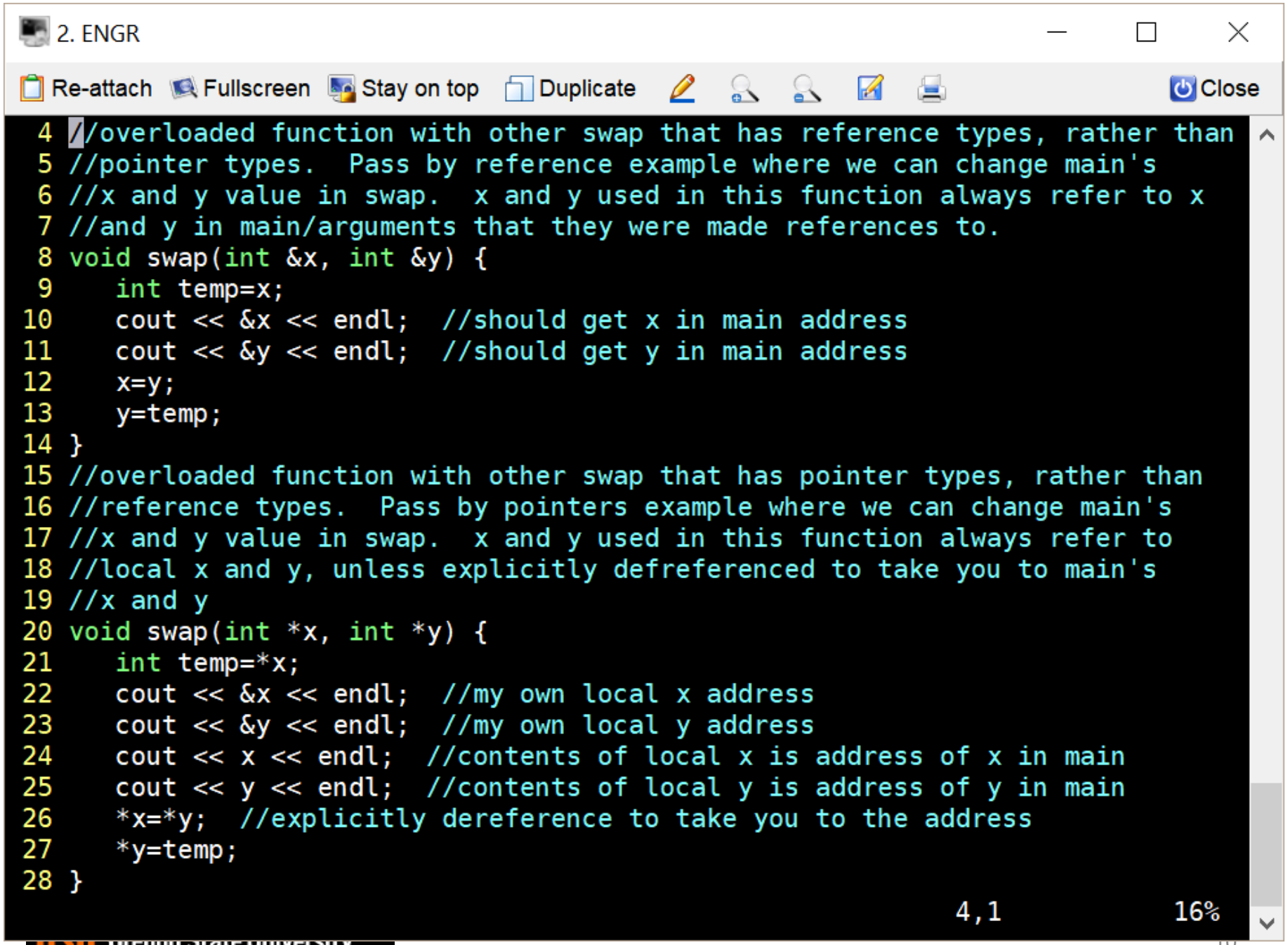
int *i;

C/C++ Pointers

```
void swap(int *, int *);
int main() {
   int a=5, b=10;
   swap(&a, &b);
   cout << "a: " << a << "b: " << b;
}
void swap(int *x, int *y) {
   int temp = *x;
   *x = *y;
   *y = temp;
}
```




2. ENGR
Re-attach
Fullscreen
Stay on top
Duplicate
Close
1 #include <iostream>
2
3 using namespace std;
4
5 //overloaded functions with different parameter types
6 void swap(int &, int &);
7 void swap(int *, int *);
8
9 int main() {
10 int x=3, y=2;
11
12 cout << &x << endl; //gives us address of where main's x is memory
13 cout << &y << endl; //gives us address of where main's y is memory
14
15 swap(x,y); //call swap with reference paramters
16 cout << x << " " << y << endl;
17
18 swap(&x,&y);//call swap with pointer parameters, explicitly pass address
19 cout << x << " " << y << endl;
20
21 return 0;
22 }
-- INSERT --
1,20
Top


2. ENGR
Re-attach
Fullscreen
Stay on top
Duplicate
Close
4 //overloaded function with other swap that has reference types, rather than
5 //pointer types. Pass by reference example where we can change main's
6 //x and y value in swap. x and y used in this function always refer to x
7 //and y in main/arguments that they were made references to.
8 void swap(int &x, int &y) {
9   int temp=x;
10   cout << &x << endl; //should get x in main address
11   cout << &y << endl; //should get y in main address
12   x=y;
13   y=temp;
14 }
15 //overloaded function with other swap that has pointer types, rather than
16 //reference types. Pass by pointers example where we can change main's
17 //x and y value in swap. x and y used in this function always refer to
18 //local x and y, unless explicitly defreferenced to take you to main's
19 //x and y
20 void swap(int *x, int *y) {
21   int temp=*x;
22   cout << &x << endl; //my own local x address
23   cout << &y << endl; //my own local y address
24   cout << x << endl; //contents of local x is address of x in main
25   cout << y << endl; //contents of local y is address of y in main
26   *x=*y; //explicitly dereference to take you to the address
27   *y=temp;
28 }
4,1
16%

Pointer and References Cheat Sheet

- *
  - If used **in a declaration** (which includes function parameters), it **creates** the pointer.
    - Ex. int *p; //p will hold an address to where an int is stored
  - If used **outside a declaration**, it **dereferences** the pointer
    - Ex. *p = 3; //**goes to the address** stored in p and stores a value
    - Ex. cout << *p; //**goes to the address** stored in p and fetches the value
- &
  - If used **in a declaration** (which includes function parameters), it **creates and initializes** the reference.
    - Ex. void fun(int &p); //p will refer to an argument that is an int by implicitly using *p (dereference) for p
    - Ex. int &p=a; //p will refer to an int, a, by implicitly using *p for p
  - If used **outside a declaration**, it means **"address of"**
    - Ex. p=&a; //**fetches the address of** a (only used as rvalue!!!) and store the address in p.