Variables vs. Pointers

• Value Semantics
  – Values stored directly
  – Copy of value is passed
  ```c
  int i, j=2;
i=j;
  ```

• Pointer Semantics
  – Address to variable is stored
  – Copy of address is passed
  ```c
  int *i, j=2;
i=&j;
  ```
# C++ Program Example

```c++
int main() {
    int x = 10, y = 20;
    int &r = x;
    int *p = 0;
    cout << "x lives: " << &x << endl;
    cout << "r lives: " << &r << endl;
    cout << "r contents: " << r << endl;
    r = y;
    cout << "x contents: " << x << endl;
    p = &y;
    cout << "p lives: " << &p << endl;
    cout << "contents of p: " << p << endl;
    cout << "y lives: " << &y << endl;
    cout << "contents of what p points to: " << *p << endl;
    *p += 5;
    cout << "contents of y: " << *p << endl;
    return 0;
}
```
//overloading functions only happens with different number of params or
//different types of params with a function of the same name

void fun(int &f) {
    f=30; //since f refers to x in main, I can change x in main
    cout << "fun" << endl;
}

int fun(int *f) {
    *f=50; //since f points to x in main, I can dereference f to get to x
    cout << "fun1" << endl;
    return 0;
}

int main() {
    int x=10, y=20;

    int &r=x;
    int *p=0;

    fun(x); //because I am passing a reference it calls 1st fun
    cout << "x" << x << endl;
    fun(&x); //because I am passing an address it calls 2nd fun
    cout << "x" << x << endl;

    cout << "x lives: " << &x << endl;
}
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.