Pointers

What is a Pointer?

Address	contents
0	01010011
1	01110101
2	01110011
3	01100001
4	01101110
	⋮
n–1	00000011

- Memory consist of bytes, each having an address.
- Address: **lvalue**, and
- Actual Value: **rvalue**.
- Compiler gets two things from a variable declaration:
  - **type** and **name**,
  - Using type sets aside a block of memory (4 bytes for `int`)
  - Enters name in symbol table: for relative memory address of the block set aside.

Pointers

What is a Pointer?

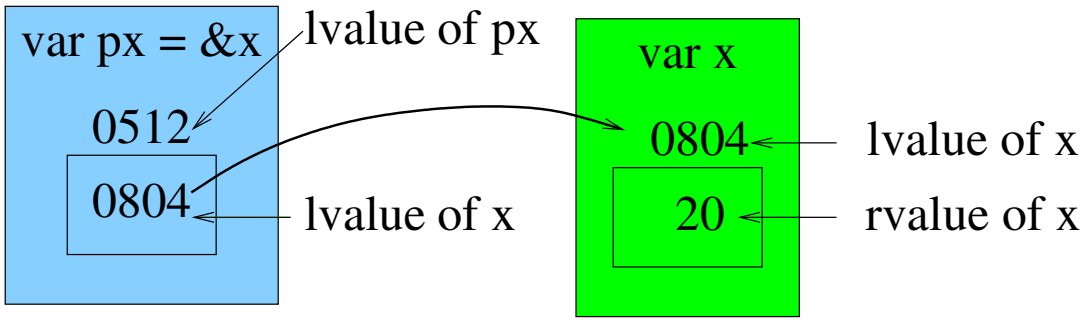
- Consider the following example:

```
L1:  int i, j, k;
L2:  j = 20;
L3:  k = j + 5;
```

- At L2, compiler interprets `j` as **lvalue**, and generates code to copy 20 to that address.
- At L3, compiler interprets `j` as **rvalue** which is the value stored at the memory location set aside for `j`.

Pointers

What is a Pointer?



- Variable holding a **lvalue** of another variable is known as pointer variable.
- By referring to that **lvalue**, stored **rvalue** can be obtained.

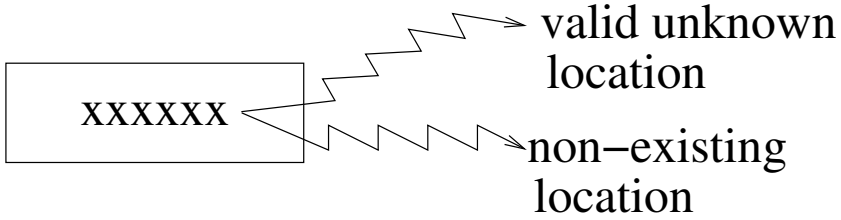
Pointers

Pointer Variable

- A pointer variable is declared by its type, and prefixing `*` to its name.
- De-referencing a pointer using "`*`" fetches the **contents** of **rvalue** of the pointer.
  - Eg. `int *ip` means `ip` stores the **lvalue** of an object that can hold an integer.
  - `*ip = 7;` copies 7 in the location pointed to by `ip`.
- If `ip = &k;` then above assignment will change **rvalue** of `k` to 7.
- The **rvalue** can be printed by de-referencing `ip`, i.e., `printf("%d\n" *ip)` prints 7.

Pointers

- When `int i;` declared outside a function, `i` is initialized to 0.
- Likewise, when `ip` is outside a function, then `ip` is initialized to a null pointer (`(nil)`).
- Otherwise, de-referencing uninitialized pointer variable is dangerous.




```
#include <stdio.h>
int *ip;
int i;
int main() {
    printf("p = %p    i = %d\n", ip, i);
}
```

Pointers as Arguments

Implications of Pass by Value

- Arguments are protected against modification.
- Simple functions like decomposing a real value into its **integral** and **fractional** is not implementable, because:

```
void decompose(double x, int xw, double xf) {
     xw = x;          // xw now has integral part
     xf = x - xw;     // xf now has fractional part
}
```

- Old values are restored back on return.
- Then how `decompose` can be written?

Pointers as Arguments

Implications of Pass by Value

- Using pointer, **lvalue** of `x` or `&x` can be passed.
- Though C uses pass by value, `&x` is copied into the matching local pointer variable.
- So, changes made through local pointer variable, modifies contents of `x` in the caller.
- The above method of communication is referred to **pass by reference**.

```
void decompose(double x, int *xw, double *xf) {
     *xw = x;          // *xw now has integral part
     *xf = x - *xw;    // *xf now has fractional part
}
```