04 - Lecture - pointers

Reading
  - Chapters 5,6

Pointers and addresses

A pointer is a variable that holds memory addresses:

```
    int x = 1, y = 2;

    int *p; // p is a pointer variable

    p = &x; //   that holds an address of an int variable

    y = *p; // y is now 1

    *p = 0; // x is now 0.  Note that *p is an l-value.

    p = &y; // p now points to y

    *p = 2; // y is now 2, x is still 0

    ++*p;   // y is now 3

    (*p)++; // y is now 4.  Note that * and ++ go right-to-left.
```

A pointer is typed:

```
    int      i = 1234;
    double   d = 3.14;

    int    *pi = &i;
    double *pd = &d;

    pi = pd;  // compiler error

    pi = (int *)pd;  // compiles, but you better know what you’re doing...

    void *pv;

    pv = pi;  // void pointer can hold any type of pointer

    i = *pv;  // compiler error - can’t dereference a void pointer

    i = *(int *)pv;
    pi = (int *)pv;  // you get back the int pointer by casting
```

NULL pointer:

  - pointer that holds the special memory address: 0

  - it is a runtime error to dereference it - segmentation fault

  - used to initialize a pointer variable:

```
        int    *pi = 0;
        double *pd = NULL;  // C/C++ provides: #define NULL 0
```

  - a pointer can turn into a boolean, just like an integer can:
    a NULL pointer is false, and every other pointer is true.

  - don’t confuse a NULL pointer with a pointer to a variable that
    holds 0:

```
        char c = 0;
        char *p = &c;

        if (p) { // true

        if (*p) { // false

        char *q = 0;

        if (q) { // false

        if (*q) { // crash!
```

Simulating call-by-reference using pointers

C is "call-by-value" language

  - function parameters are passed by value, i.e., by *copying*

  - there is NO WAY to do call-by-reference in C (you can do it in C++)

```
        void swap(int x, int y) // WRONG
        {
            int temp;
            temp = x;
            x = y;
            y = temp;
        }

        int x = 1, y = 2;
        swap(x, y);
        // didn’t work: x is still 1, y is still 2
```

  - But you can *fake* it using pointers:

```
        void swap(int *px, int *py)
        {
            int temp;
            temp = *px;
            *px = *py;
            *py = temp;
        }

        int x = 1, y = 2;
        swap(&x, &y);
        // now x is 2 and y is 1
```

  - Note that it’s still call-by-value: the addresses are passed by value.