```c
int x = 3; int y = 7; int z = 5;  // !@#
x = y++ * ++ z

<table>
<thead>
<tr>
<th>expression</th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>x = y++ * ++ z</td>
<td>3</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>x = 7 * ++ z</td>
<td>3</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>x = 7 * 6</td>
<td>3</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>x = 42</td>
<td>3</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>42</td>
<td>42</td>
<td>8</td>
<td>6</td>
</tr>
</tbody>
</table>
```
Pointer Operations

For every type there is a pointer type:
- `int *`  *pointer to an int*
- `double *`  *pointer to a double*
- `char *`  *pointer to an char, etc.*

`pointer` == `address` == `reference`  in C

(not all addresses are legal for all data types)
int i = 7;
int *iptr;
iptr = &i;
scanf("%d", iptr);    //scanf("%d", &i);
printf("%d", *iptr);  //printf("%d", i);

As a prefix, * is the dereferencing operator.
(dereferencing is “pointer chasing”)
--> -->
#include <stdio.h>

void swap(int *, int *);

int main()
{
    int x = 3;
    int y = 5;
    printf("Before the function: x = %d y = %d\n", x, y);
    swap(&x, &y);
    printf("After the function: x = %d y = %d\n", x, y);
    return 0;
}

void swap(int *a, int *b)
{
    int temp;
    printf("Before the exchange: a = %d b = %d\n", *a, *b);
    temp = *a;
    *a = *b;
    *b = temp;
    printf("After the exchange: a = %d b = %d\n", *a, *b);
}
When to use pass by address?

I prefer *When to pass an address?*

When more than one value needs to be returned to the calling function. AND

The function continues to perform a single fundamental task. *why this?*