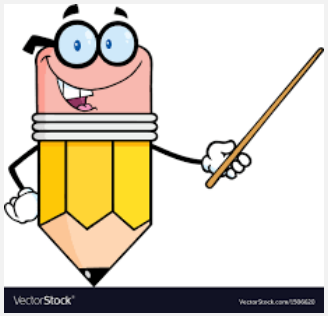
Pointers Review



Pointers in C: Review (1)

- C's integration of pointers, arrays and address arithmetic is one of the strengths of the language
- In general, a pointer can be initialized just as any other variable can, though normally the only meaningful values are zero or an expression involving the addresses of previously defined data of appropriate type.

```
char buffer[MAXSIZE];
char *p = buffer; // or  char *ptr = &buffer[0]
```

- If `p` is a pointer to some element of an array, then `p++` increments `p` to point to the next element, and `p += i` increments it to point `i` elements beyond where it currently does.
- The address scales correctly based on the size of the type. So `p++` for a pointer to `char` will increment by 1, a pointer to `int` will increment by `sizeof(int)`, a pointer to a `double` will increment by `sizeof(double)` and so on (including `structs`)

Pointers in C: Review (2)

- Pointers and integers are not interchangeable. Zero is the sole exception: the constant zero may be assigned to a pointer, and a pointer may be compared to the constant zero. Zero is never a valid address in C. The header file `<stdio.h>` defines NULL as zero.
- Pointer may be compared if they point to the same type. Any pointer can be meaningfully compared for equality and inequality with zero. However, the behavior is undefined for comparisons with pointers that do not point to members of the same array or for arithmetic in comparisons.
- Pointer subtraction is also valid if the two pointers point to elements of the same array. For example, if `p < q`, then `q - p + 1` is the number of elements from `p` to `q` inclusive.

Pointers in C: Review (3)

- Valid pointer operations:
  - assignment of pointers of the same type,
  - adding or subtracting a pointer and an integer,
  - subtracting or comparing two pointers to members of the same array,
  - and assigning or comparing to zero (aka `NULL`)
- All other pointer arithmetic is illegal! For example:
  - adding two pointers
  - multiply, divide, shift or mask pointers
  - add a float or a double value to a pointer
  - assign a pointer of one type to a pointer of another type without a cast (except for void *)

Pointers in C: Exercises

- Consider the following code:

```
struct test {
    int x;
    int y;
    char *s;
};
const int n = 100;
struct test *array = (struct test *) malloc(sizeof(struct test)*n);
void *ptr = array;
```

  Based on the above code, which of the following expressions correctly access the value stored in `array[i]`, where `i` $<$ `n`.

  - `*(ptr + i)`
  - `*(struct test *)(ptr + i)`
  - `*(ptr + 12*i)`
  - `*((struct test *)ptr + i)`