Pointer arithmetic – arrays only

- Can add or subtract an integer – as long as result is still within the bounds of the array
- Can subtract a pointer from another pointer – iff both point to elements of the same array

```c
char word[] = "cat";
    /* create array of four chars: ‘c’ ‘a’ ‘t’ ‘\0’ */
char *p = word; /* point p at first char */
while (*p++ != ‘\0’); /* move pointer to end */
printf("word length: %d", p-word-1);
    /* subtract one address from another – result is 3 */
```

- But – no pointer multiplication or division, and cannot add two pointers
void stringcopy(char *s, char *t)

- One way to implement – use subscript notation:
  ```c
  int i = 0;
  while ((s[i] = t[i]) != '\0') i++;
  ```

- Another way – use the pointer parameters:
  ```c
  while ((*s = *t) != '\0')
  {
    s++; t++;
  }
  ```

- Usually just increment in the while header:
  ```c
  while ((*s++ = *t++) != '\0');
  ```

- And it’s possible to be even more cryptic:
  ```c
  while (*s++ = *t++); /* Actually works! */
  ```
Multi-dimensional and pointer arrays, and pointers to arrays

- **Multi-dimensional arrays** – arrays of arrays
  - `int x[5][3]; /* allocates memory for 15 ints */`
  - Actually, 5 arrays, each able to store 3 integers

- **Arrays of pointers**
  - `int *p[5]; /* allocates memory for 5 pointers */`
    - `for (i=0; i<5; i++) p[i] = x[i]; /* x as above */`
    - Now p can be used as an alias for x

- **Pointers to arrays** – require pointers to pointers
  - `int **px = x; /* points to first array in x */`
  - `px++; /* moves pointer to next array */`
Command line arguments

- Declare main with two parameters
  - An argument count, and an array of argument values
    ```c
    int main(int argc, char *argv[]) {...}
    ```
  - `argc = 1` plus the number of tokens typed by the user at the command line after the program name
  - `argv[0]` is the program name
  - `argv[1]...[argc-1]` are the other tokens
    - Each one points to an array of characters (i.e., a C string)

- Note equivalent way to declare second parameter
  - `char **argv` – commonly used instead of above form
    - Can still use array notation, but also can `argv++` and so on
sizeof

- A unary operator – computes the size, in bytes, of any object or type
  - Usage: `sizeof object` or `sizeof(type)`
  - If `x` is an `int`, `sizeof x == sizeof(int)` is true
- Works for arrays too – total bytes in whole array
  - Sometimes can use to find an array’s length:
    ```
    int size = sizeof x / sizeof x[i];
    ```
- Actual type of result is `size_t`
  - An unsigned integer defined in `<stddef.h>`
  - Similarly, `diff_t` is result type of pointer subtraction
- Especially useful to find the sizes of structures
C structures

- Structures are variables with multiple data fields
- e.g., define structure to hold an `int` and a `double`:
  ```c
  struct example{
    int x;
    double d;
  };
  ```
- Create a structure, and assign a pointer to it
  ```c
  struct example e, *ep = &e;
  ```
- Now can access fields by `e` or by `ep`:
  ```c
  e.d = 2.5; /* use name and the dot `.` operator */
  ep->x = 7; /* or use pointer-to-structure-field `->` operator */
  ```
  - Second way is short-cut version of: `(*ep).x = 7;`
- **Note:** `sizeof e >= sizeof(int)+sizeof(double)`
typedef and macros

● Can precede any declaration with typedef
  – Defines a name for the given type:
    
    ```c
    typedef struct example ExampleType;
    ExampleType e, *ep; /* e, ep same as prior slide */
    
    – Very handy for pointer types too:
    ```
    ```c
    typedef ExampleType *ETPointer;
    ETPointer ep; /* ep same as above */
    ```

● Macros can simplify code too
  ```c
  #define X(p) (p)->x
  X(ep) = 8; /* preprocessor substitutes correct code */
  ```
Unions

- Can hold different data types/sizes (at different times)
- e.g., define union to hold an int or a double:
  ```c
  union myValue{
    int x;
    double d;
  } u, *up; /* u is a union, up can point to one */
  ```
- Access x or d by u. or up-> just like structures
- sizeof u is size of largest field in union
  - Equals sizeof(double) in this case
- Often store inside a structure, with a key to identify type
And see:

~mikec/cs12/demo01/*.c