

# CMPSC 311 - Introduction to Systems Programming

More on Pointers and Arrays

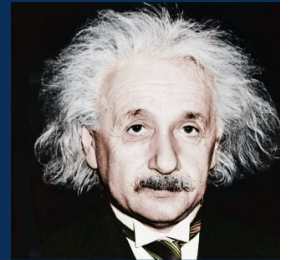
Professors:  
Suman Saha

(Slides are mostly by *Professor Patrick McDaniel*  
and *Professor Abutalib Aghayev*)

**Pointers**



**Pointers to  
pointers**



**Pointers to  
pointers to  
pointers**



**Pointers to  
pointers to  
pointers to  
pointers**



# Multidimensional Array Layout



- Arrays are laid out in row-major order
- `int x[2][4] = {{1, 2, 3, 4}, {5, 6, 7, 8}};`

1	2	3	4	5	6	7	8
---	---	---	---	---	---	---	---

- `int x[3][2][4] = {{{1, 2, 3, 4}, {5, 6, 7, 8}},  
                  {{9, 10, 11, 12}, {13, 14, 15, 16}},  
                  {{17, 18, 19, 20}, {21, 22, 23, 24}}};`

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

- Multidimensional arrays must have bounds for all dimensions except first
  - above, the followings are okay: `int x[][4] = ...;` `int x[][2][4] = ...;`
  - given an expression `x[1][2]`, we need the length of the second row to compute offset

# Arrays of Strings



```
char planets[][8] = {
```

```
    "Mercury",
```

```
    "Venus",
```

```
    "Earth",
```

```
    "Mars",
```

```
    "Jupiter",
```

```
    "Saturn",
```

```
    "Uranus",
```

```
    "Neptune"
```

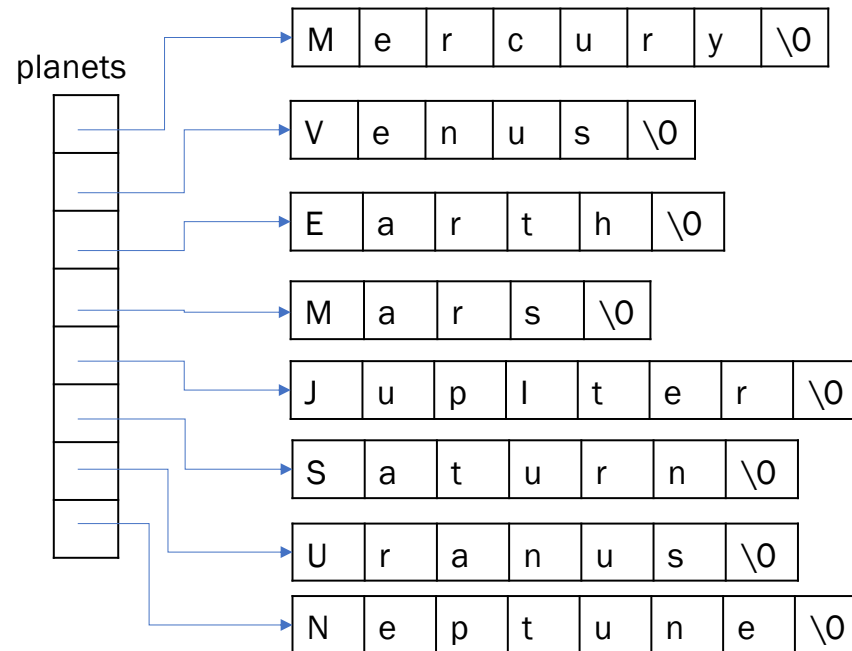
```
};
```

M	e	r	c	u	r	y	\0
V	e	n	u	s	\0	\0	\0
E	a	r	t	h	\0	\0	\0
M	a	r	s	\0	\0	\0	\0
J	u	p	i	t	e	r	\0
S	a	t	u	r	n	\0	\0
U	r	a	n	u	s	\0	\0
N	e	p	t	u	n	e	\0

# Arrays of Strings



```
char *planets[] = {  
    "Mercury",  
    "Venus",  
    "Earth",  
    "Mars",  
    "Jupiter",  
    "Saturn",  
    "Uranus",  
    "Neptune"  
};
```



# Arrays of Strings



PennState

```
char *planets[] = {
```

```
    "Mercury",
```

```
    "Venus",
```

```
    "Earth",
```

```
    "Mars",
```

```
    "Jupiter",
```

```
    "Saturn",
```

```
    "Uranus",
```

```
    "Neptune"
```

```
};
```

```
    for (int i = 0; i < 8; ++i)
        if (planets[i][0] == 'M')
            printf("%s\n", planets[i]);
```

```
    for (char **p = planets; p < planets + 8; ++p)
        if (**p == 'M')
            printf("%s\n", *p);
```

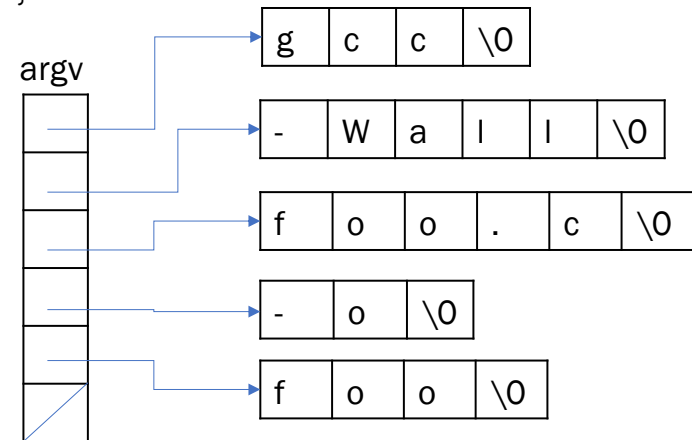
# Command-line arguments



- So far, our main functions was accepting no (void) parameters:
  - `int main(void) { ... }`
- We can access command-line arguments using following parameters:
  - `int main(int argc, char *argv[]) { ... }`
- Since arrays decay into pointers, the following is OK too:
  - `int main(int argc, char **argv) { ... }`

```
$ gcc -Wall foo.c -o foo
```

```
int main(int argc, char *argv[]) {  
    for (int i = 0; i < argc; ++i)  
        printf("%s\n", argv[i]);  
    for (char **p = argv; *p != NULL; p++)  
        printf("%s\n", *p);  
}
```



# Pointers to Pointers



- We already know how to modify non-pointer variables through functions:
  - we pass their address to the function, i.e. we pass pointers to the variables

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

```
void swap(int *x, int *y) { int tmp = *x; *x = *y; *y = tmp; };
```

- What if we want to modify a pointer variable itself?

```
int x = 1, y = 2, *xp = &x, *yp = &y;
```

```
void swapception(int **xp, int **yp) { int *tmp = *xp; *xp = *yp; *yp = tmp; };
```

- Why? Data structures! (linked lists, trees, etc.)



# Pointers to Pointers to Pointers

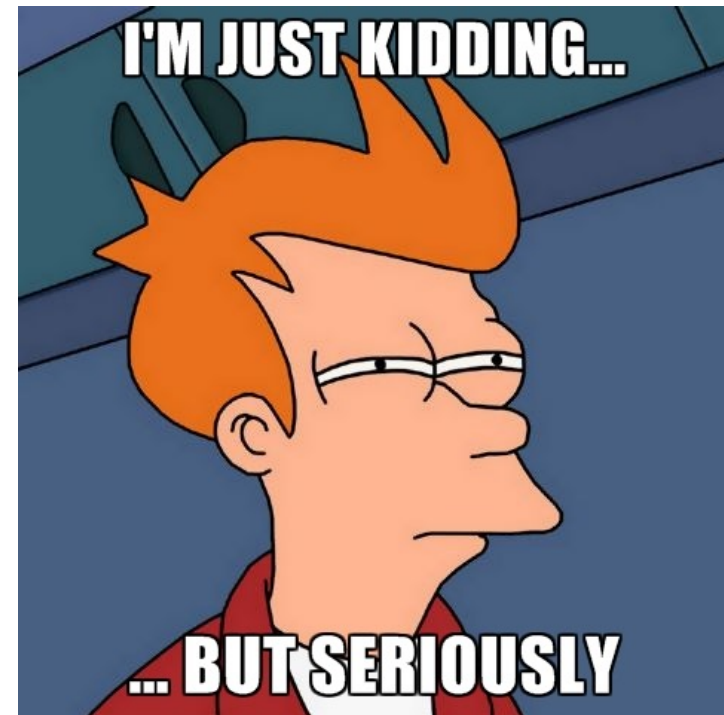


```
#include <stdio.h>

char *c[] = {
    "ENTER",
    "NEW",
    "POINT",
    "FIRST"
};

char **cp[] = {c+3, c+2, c+1, c};
char ***cpp = cp;

int main() {
    printf("%s", **++cpp);
    printf("%s ", *--*++cpp+3);
    printf("%s", *cpp[-2]+3);
    printf("%s\n", cpp[-1][-1]+1);
}
```





# Pointers to Functions



```
#include <stdio.h>
#include <stdlib.h>

// void qsort(void *base, size_t nmemb, size_t size, int (*compar)(const void *, const void *));

int values[] = { 88, 56, 100, 2, 25 };

int cmpfunc (const void * a, const void * b) {
    return *(int*)a - *(int*)b;
}

int main (void) {
    qsort(values, 5, sizeof(int), cmpfunc);
    return 0;
}
```

# Prelude to Assignment 3



**Edsger Dijkstra**

**Program testing can be used to show the presence of bugs, but never to show their absence.**

# Prelude to Assignment 3



Alan Perlis

Fools Ignore complexity. Pragmatists suffer it. Some can avoid it. Geniuses remove it.

Simplicity does not precede complexity, but follows it