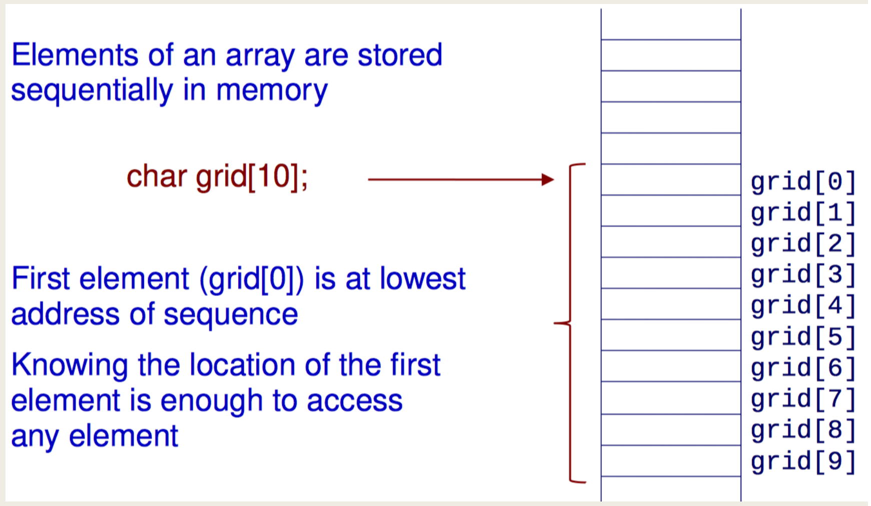
CS 211 RECITATIONS WEEK 3

Wenjie Qiu
Teaching Assistant
Office Hour: Thursday noon – 1pm
wenjie.qiu@rutgers.edu
https://github.com/Roadsong/rutgers-cs211-recitations

Content

- Data Structures
  - *Arrays & pointers & pointer arithmetic*
  - *Strings*
  - *Structures*
- Memory Management
  - *Memory Layout*
  - *Dynamic Allocation*
  - *Dangling Pointer & Segmentation Fault*
- File I/O
  - *Unix/Linux vs. C*
  - *fopen/fclose & fprintf/fscanf*
  - *stadin & stdout & stderr*
- Two issues in Jeff's slides
- Useful Resources

Data Structures - Array


Elements of an array are stored sequentially in memory
char grid[10];
First element (grid[0]) is at lowest address of sequence
Knowing the location of the first element is enough to access any element
grid[0]
grid[1]
grid[2]
grid[3]
grid[4]
grid[5]
grid[6]
grid[7]
grid[8]
grid[9]

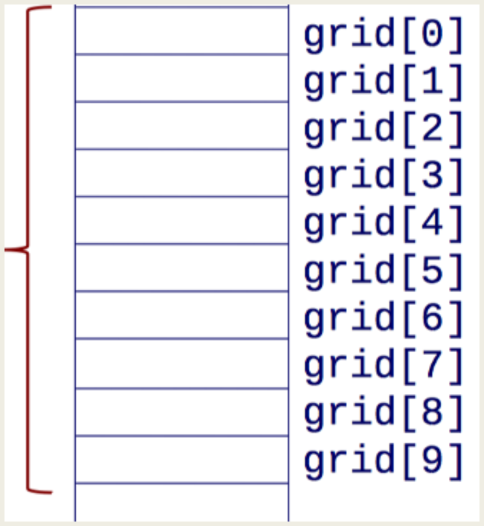
- Array are contiguous sequences of data items
  - *same type: int a[10]*
  - *access by index: a[5]*
  - *start from zero: a[0]*
- No compile time / runtime check
  - *no boundaries check (other language like Java, Python check)*
  - *can be dangerous*
  - *a[10] can be trash*
  - *or something you think you it's meaningful (e.g., zero or other values) (but…)*
  - *In general, always don't access somewhere you have no legal access.*
- Size must be determined at compile time
  - *Marcos, SIZE, MAX_LEN, ….*

Array & Pointer & Pointer Arithmetic

- An array name is essentially a pointer to the first element in the array.
- Have a look at Jeff's c-week2 slides page 4, their equivalent form.

- Pointer Arithmetic - shift by size, not by byte.
- What if grid is an integer array?
  - *int grid[10]*
  - *&(grid[2]) - &(grid[1]) = 4 bytes*
  - *When I want to move to the next item, I only need to increment 1, but I can move 4 bytes (i.e., size of the element)*

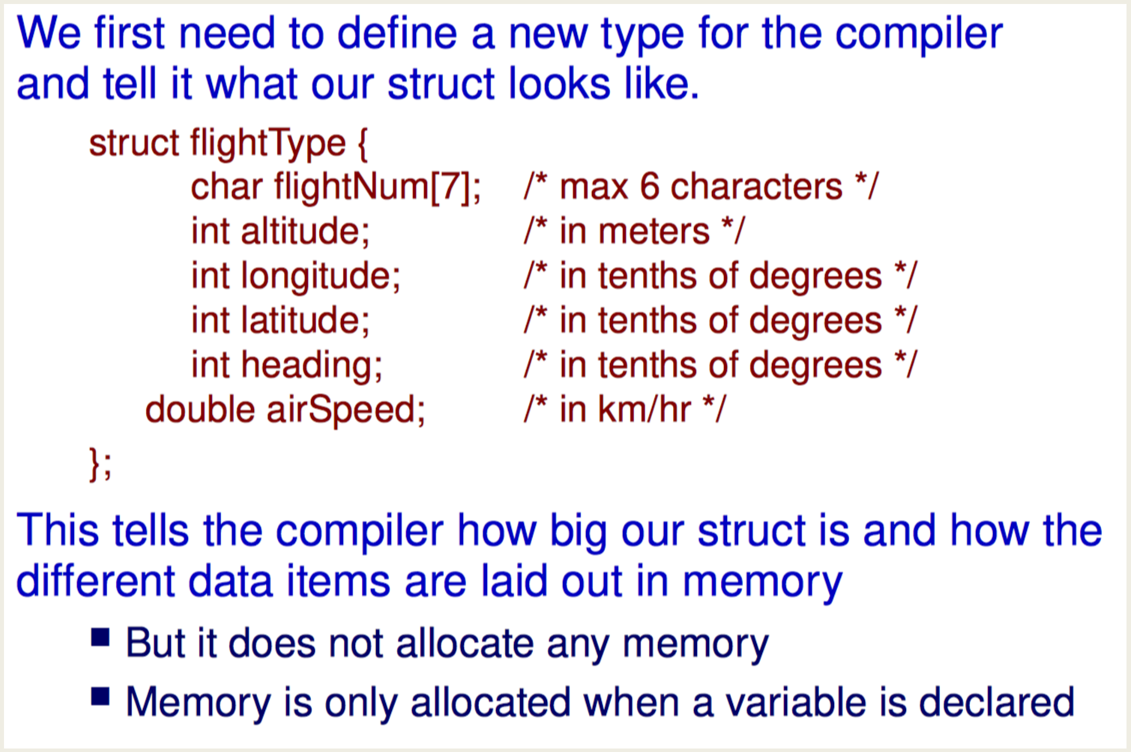
Compiler is smart!



Data Structures - String

- In C programming language, there is no formal type of string (C++, Java has string type)
  - *string mystr_1 = "hello, students!"*
- Array of characters
  - *char mystr_2[] = "hello, students!" (size can be calculated by compilers)*
  - *my can also indicate the size, if not used up, padding with '\0'*
  - *mystr_2[0] = 'H' (Yes, we can do that)*
- C string literals
  - *char *mystr_3 = "hello, students!"*
  - *mystr_3[0] = 'H' (No, segmentation fault) (But why?) (explain)*
- String libraries <string.h>
  - *Unsafe vs. safe*
  - *strcpy() / strncpy()*

Data Structures - Struct



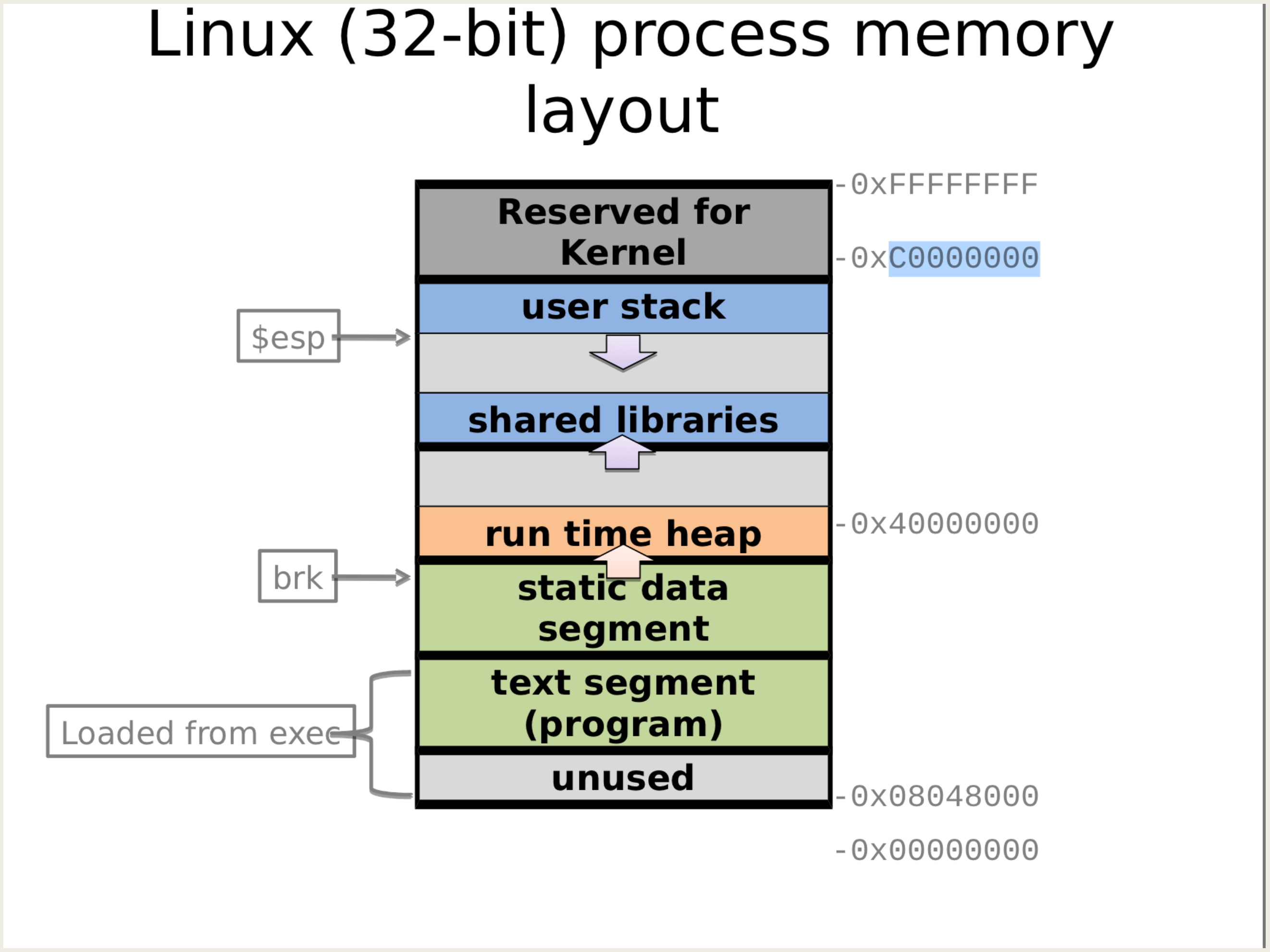
We first need to define a new type for the compiler and tell it what our struct looks like.

```
struct flightType {
        char flightNum[7];    /* max 6 characters */
        int altitude;         /* in meters */
        int longitude;        /* in tenths of degrees */
        int latitude;         /* in tenths of degrees */
        int heading;          /* in tenths of degrees */
    double airSpeed;          /* in km/hr */
};
```

This tells the compiler how big our struct is and how the different data items are laid out in memory

- But it does not allocate any memory
- Memory is only allocated when a variable is declared

What if I change char flightNum[7] to char flightNum[] ?



Virtual Memory:
1) Each process has the illusions of owning the whole memory.

2) And each process has its own virtual memory.

Benefits:
1) Memory isolation.

2) More space than physical memory.

32-bits OS: 4 GB range
64-bits OS: (48-bits range)

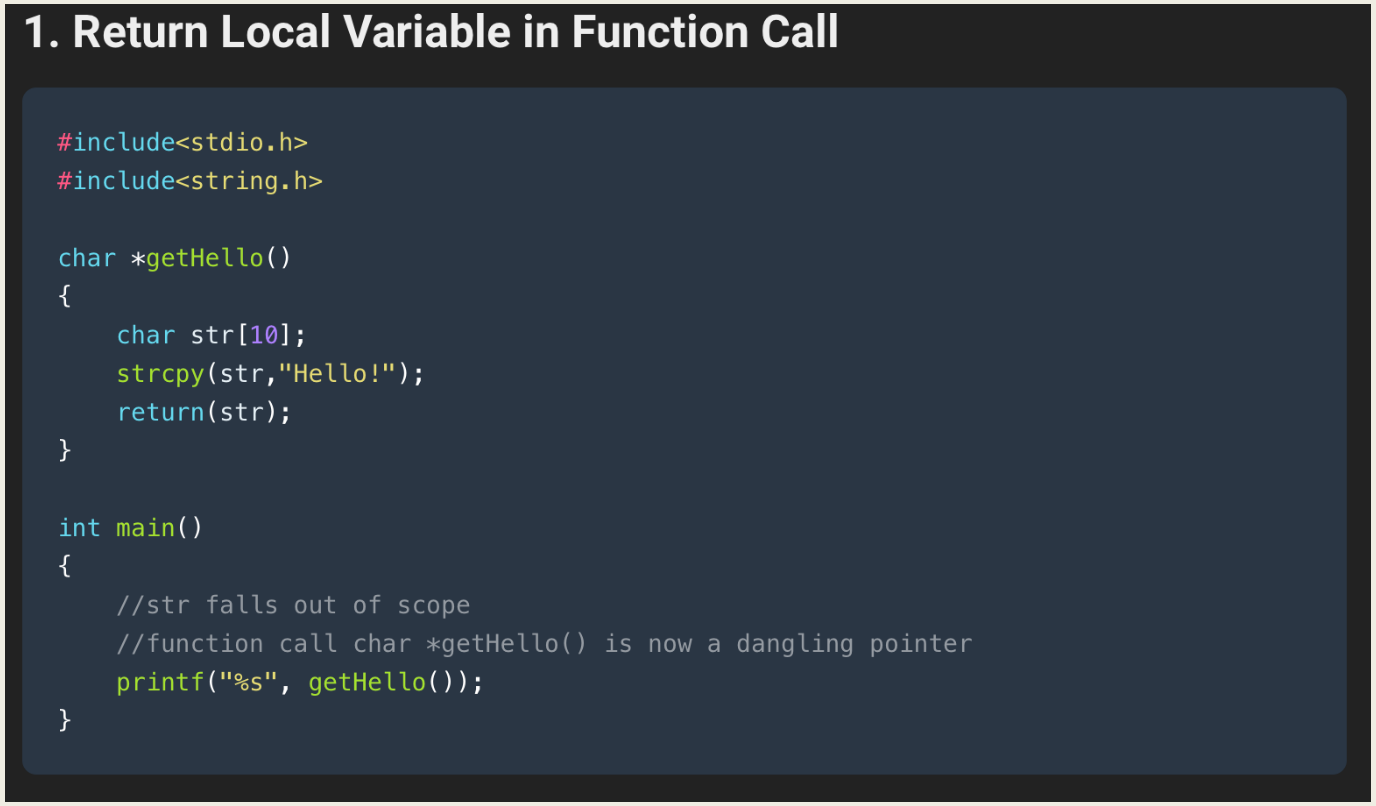
https://inst.eecs.berkeley.edu/~cs161/sp15/slides/lec3-sw-vulns.pdf

https://en.wikipedia.org/wiki/64-bit_computing

Dynamic Allocation

- Why do we need this?
  - *We want to playing with variable numbers of items.*
  - *We are tired of fixed size array, etc.*
- malloc() & free()
  - *void *malloc(size_t number_of_bytes)*
  - *Why the return type is void * ?*
  - *Data will be created in heap.*
  - *free() will reclaim the allocated memory, but the pointer still exist.*
  - *After free(q), memory that pointer q (previously) points to cannot be accessed.*
  - *Then, let q = NULL is the best practice.*

Dangling Pointer



1. Return Local Variable in Function Call

```c
#include<stdio.h>
#include<string.h>

char *getHello()
{
    char str[10];
    strcpy(str,"Hello!");
    return(str);
}

int main()
{
    //str falls out of scope
    //function call char *getHello() is now a dangling pointer
    printf("%s", getHello());
}
```

Does return (str) return the address of the array str?

- *Yes*

But why it doesn't work?

char str[10] is deleted after function getHello() finished.

https://developerinsider.co/what-is-dangling-pointer-with-cause-and-how-to-avoid-it/

Segmentation Fault

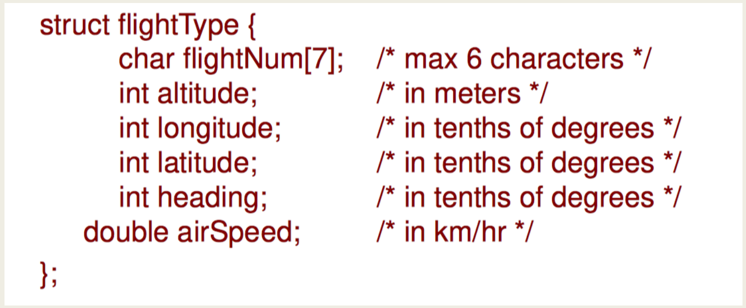
- Segmentation fault is a specific kind of error caused by accessing memory that "does not belong to you."
- Whenever you get a segmentation fault you know you are doing something wrong with memory – accessing variable that has already been freed, writing to a read-only portion of the memory, etc.
- Undefined behaviors sometimes looks good to you
  - *But don't play with it.*

https://stackoverflow.com/questions/2346806/what-is-a-segmentation-fault

File I/O

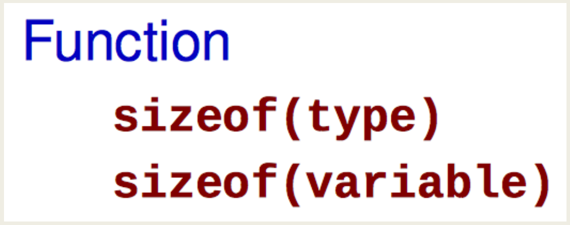
- Playing with files in a Unix/Linux way
  - *Using system calls: open(), read(), write(), close(), lseek(), …*
  - *Header files: <unistd.h> <fcntl.h> <sys/types.h>, …*
  - *They are also user space operations, but a bit complex to use.*
    - Search user space / kernel space for further information
- Playing with files in a C programming language way
  - *fopen(), fclose(), fscanf(), fprintf(), …*
  - *Header files: <stdio.h>*
  - *Higher level abstractions, extended interface to Unix/Linux file operations*
  - *We love it.*
- Standard input/output/error
  - *each executing program has its own streams for stdin, stdout, stderr*
  - *example*

Two issues in Jeff's slides



```
struct flightType {
        char flightNum[7];   /* max 6 characters */
        int altitude;        /* in meters */
        int longitude;       /* in tenths of degrees */
        int latitude;        /* in tenths of degrees */
        int heading;         /* in tenths of degrees */
    double airSpeed;         /* in km/hr */
};
```

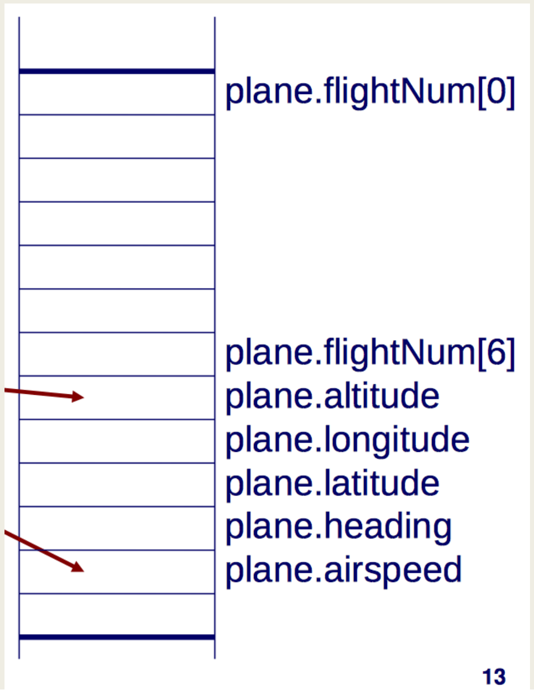
- Why is this wrong? Explain



Function

```
    sizeof(type)
    sizeof(variable)
```

- Why is this wrong? Explain 1 and 2



Useful Resources

- C Programming: A Modern Approach, 2nd Edition
  - *C89 & C99*
  - *Standard Libraries*
  - *Amazon*
- The Hardware/Software Interface
  - *University of Washington, CSE 351 (including slides & videos)*
- Hacking, The Art of Exploitation
  - *Chapter 1 Introduction & Chapter 2 Programming*
  - *Amazon*