C Tutorial for EDA092

Prajith R G

31/8/2017

Based on slides by Bhavishya Goel

Disclaimer

- This is not a full blown C tutorial
  - You cannot learn C from one lecture
- Introduction to C concepts required for EDA092, especially Lab 1
- Recommended reading material:
  - *The C Programming Language*, Kernighan, Richie
  - *C Traps and Pitfalls*", Andrew Koenig, Addison-Wesley
  - http://www.cs.cornell.edu/courses/cs414/2005sp/cforjava.pdf

Prerequisites for Lab 1

- Pointers
- String manipulation
- Recursive functions
- Linked lists

Pointers

- A “pointer” can be anything that indicates the location of an object, e.g.
  - A URL
  - A street address
  - A road sign

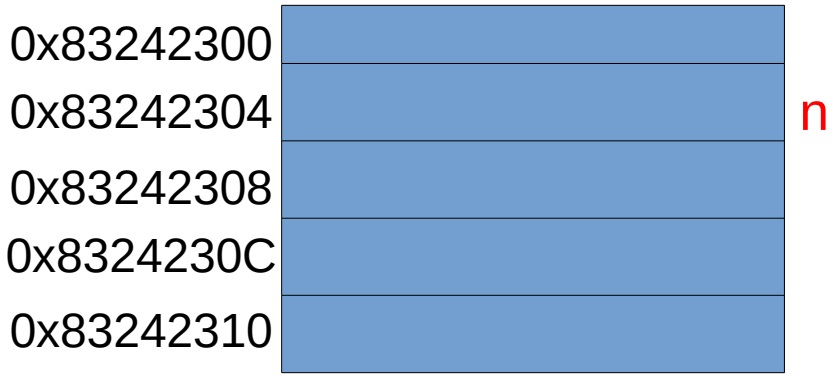
Pointers in C

- In C programming, pointers are datatypes that contain **memory addresses** of other variables

Pointers in C

- When you define a variable in C, a memory location is reserved to store contents of that variable

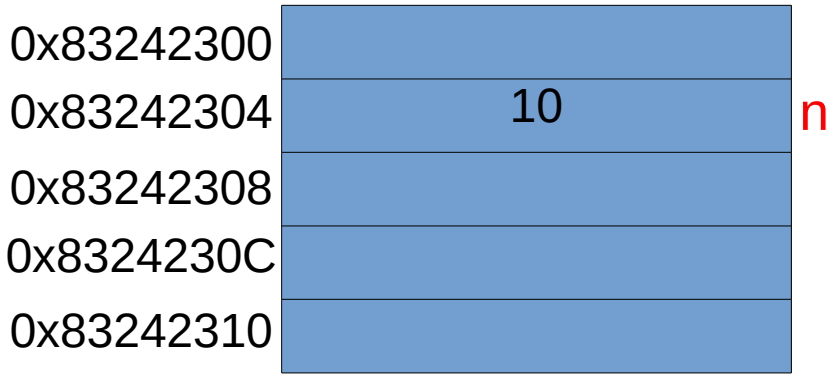
  int n;



Pointers in C

- When you define a variable in C, a memory location is reserved to store contents of that variable

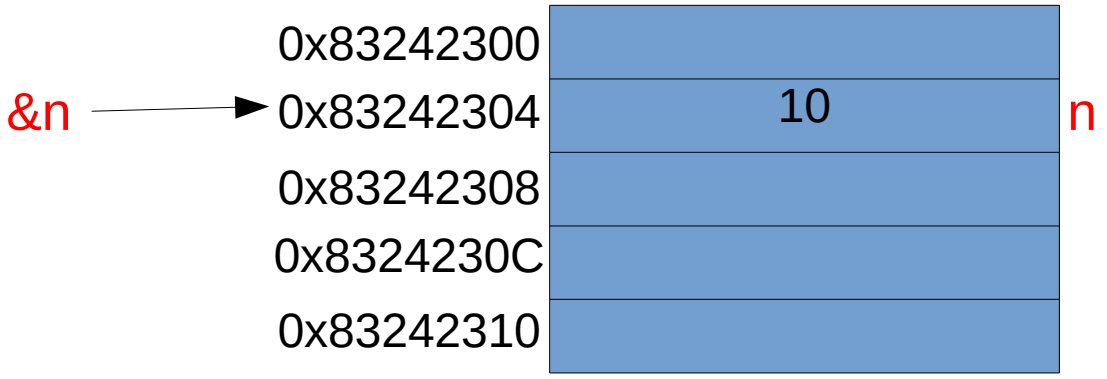
```
int n;
n = 10;
```



Pointers in C

- When you define a variable in C, a memory location is reserved to store contents of that variable

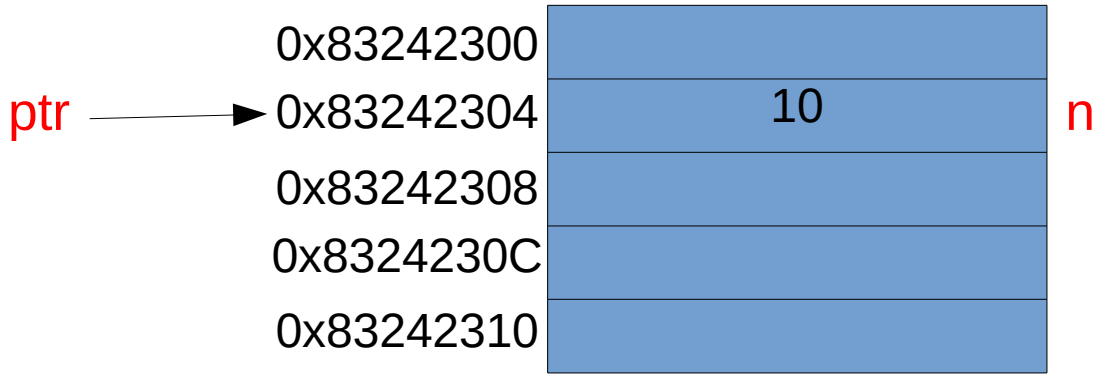
```
int n;
n = 10;
printf("address of n = %p", &n);
```



Pointers in C

- When you define a variable in C, a memory location is reserved to store contents of that variable

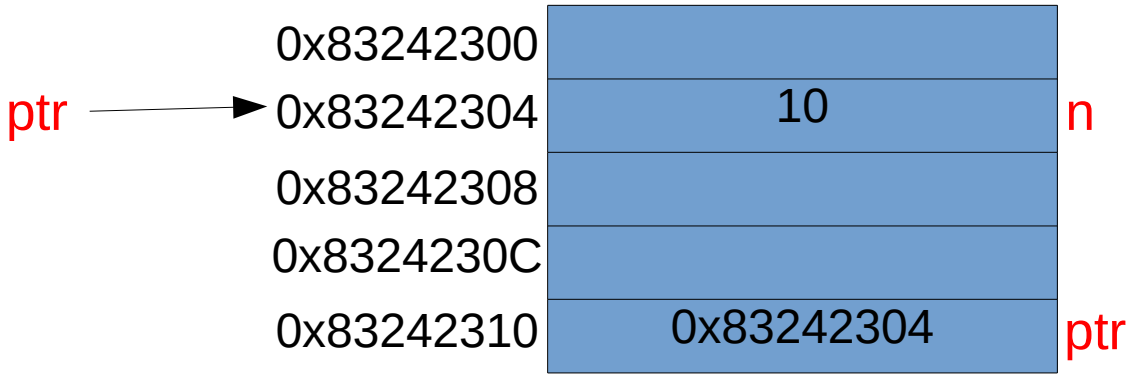
```
int n;
int* ptr = &n;
n = 10;
printf("address of n = %p", ptr);
```



Pointers in C

- When you define a variable in C, a memory location is reserved to store contents of that variable

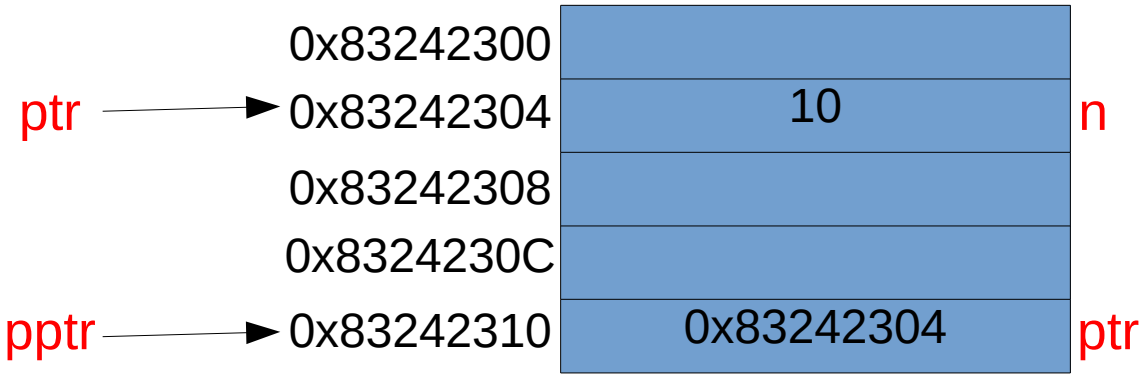
```
int n;
int* ptr = &n;
n = 10;
printf("address of n = %p", ptr);
```



Pointers in C

- When you define a variable in C, a memory location is reserved to store contents of that variable

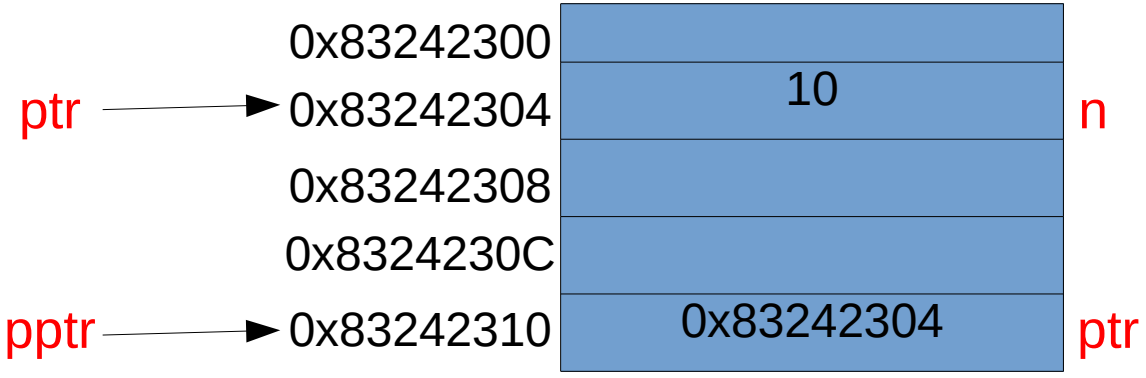
```
int n;
int* ptr = &n;
int **pptr = &ptr;
printf("address of n = %p", *pptr);
```



Pointers in C

- When you define a variable in C, a memory location is reserved to store contents of that variable

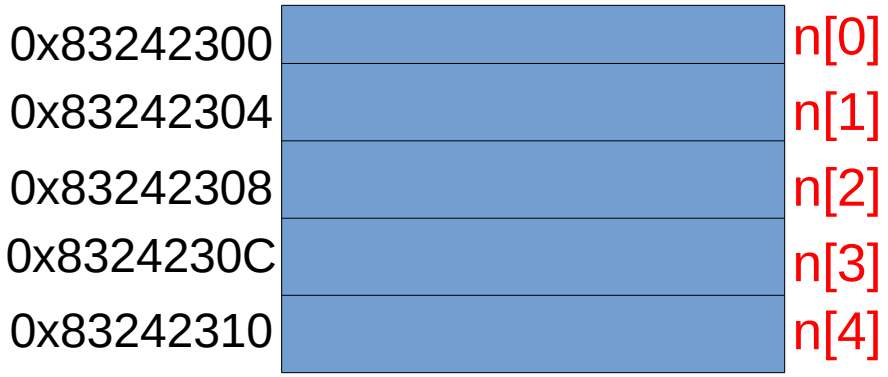
```
int n;
int* ptr = &n;
int **pptr = &ptr;
printf("address of n = %p", *pptr);
printf("value of n = %d", **pptr);
```



Array

- A fixed size sequential collection of elements of the same type

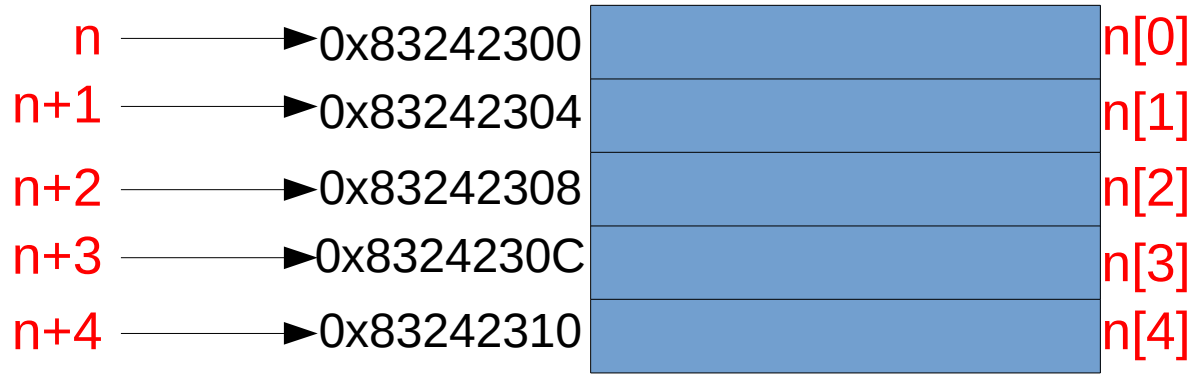
```
int n[5];
```



Array

- A fixed size sequential collection of elements of the same type
- The array name is the pointer to the first location in the array

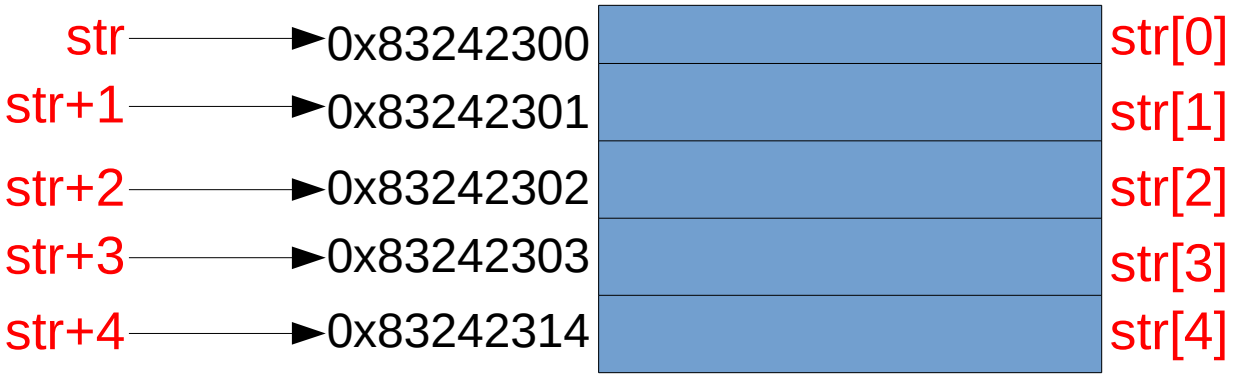
  int n[5];



String

- String in C is an array of *char* datatype

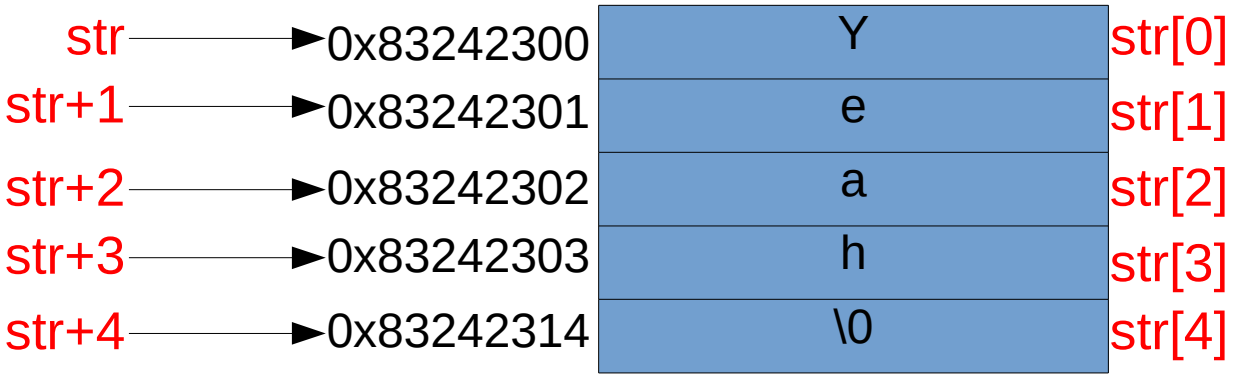
```
char str[5];
```



String

- String in C is an array of *char* datatype

char str[5] = “Yeah”;



String

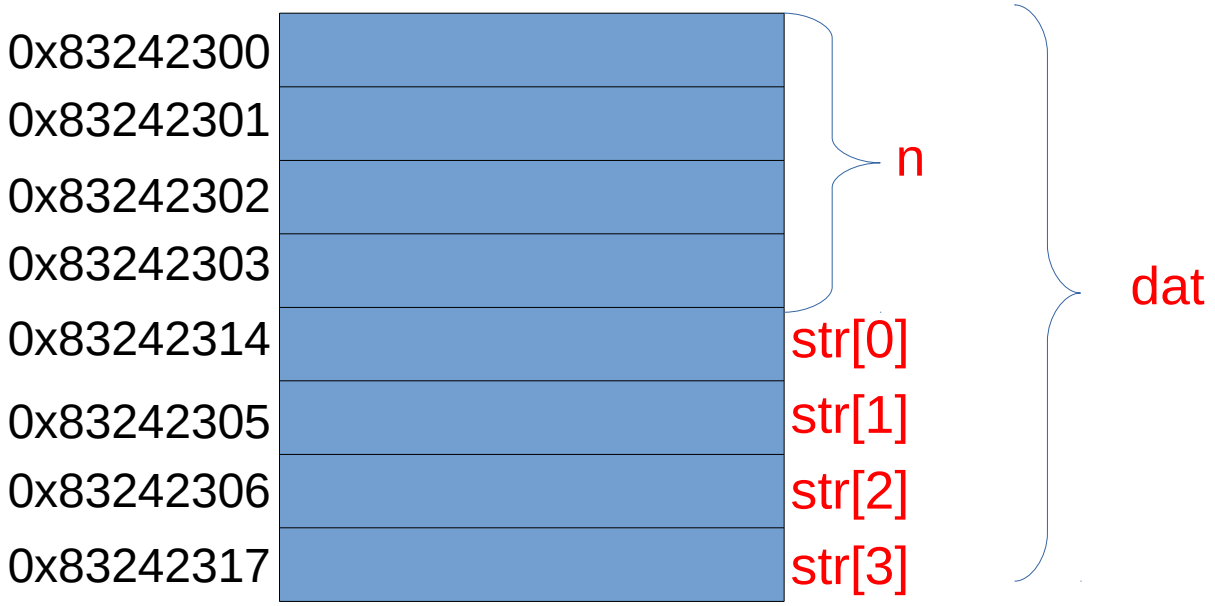
- String in C is an array of *char* datatype

```
char str[5] = “Yeah”;
char *s = str;
while(*s != '\0')
  printf (“%c\n”, *s++);
```

Output??

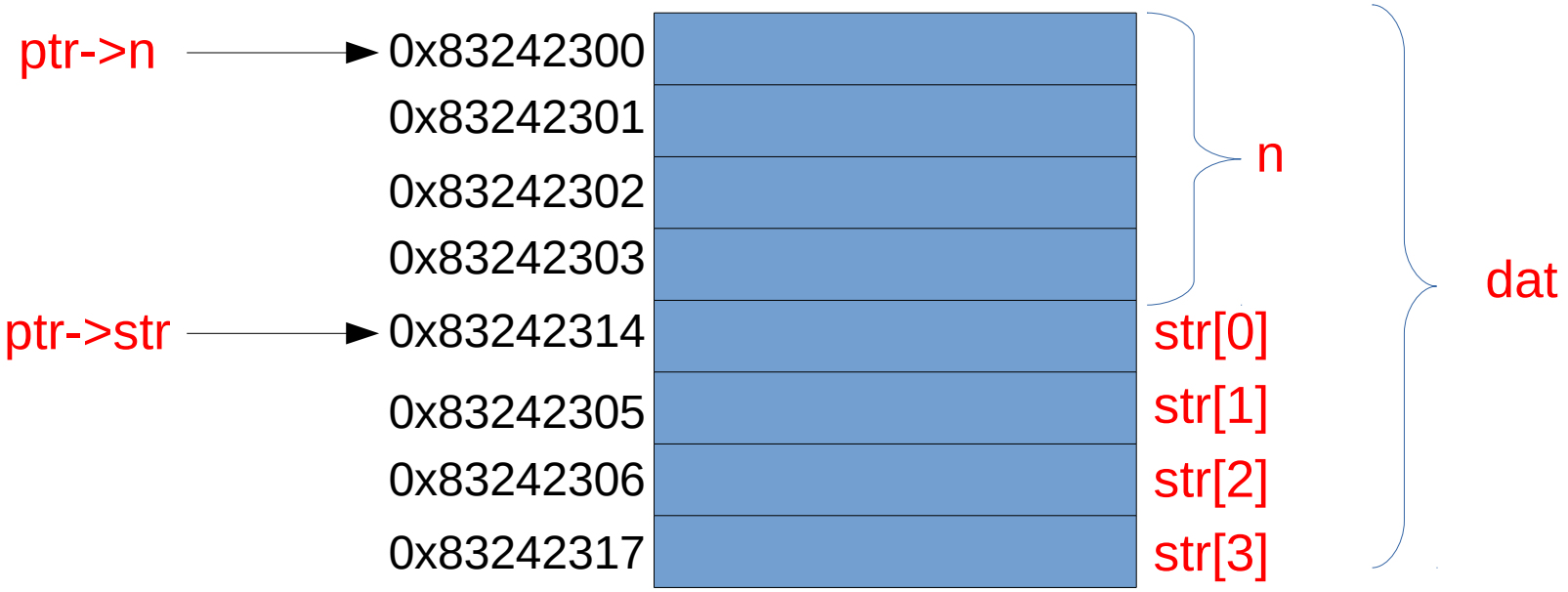
Pointers to Data Structures

```
typedef struct data {
    int n;
    char str[4];
} Data;
Data dat;
```



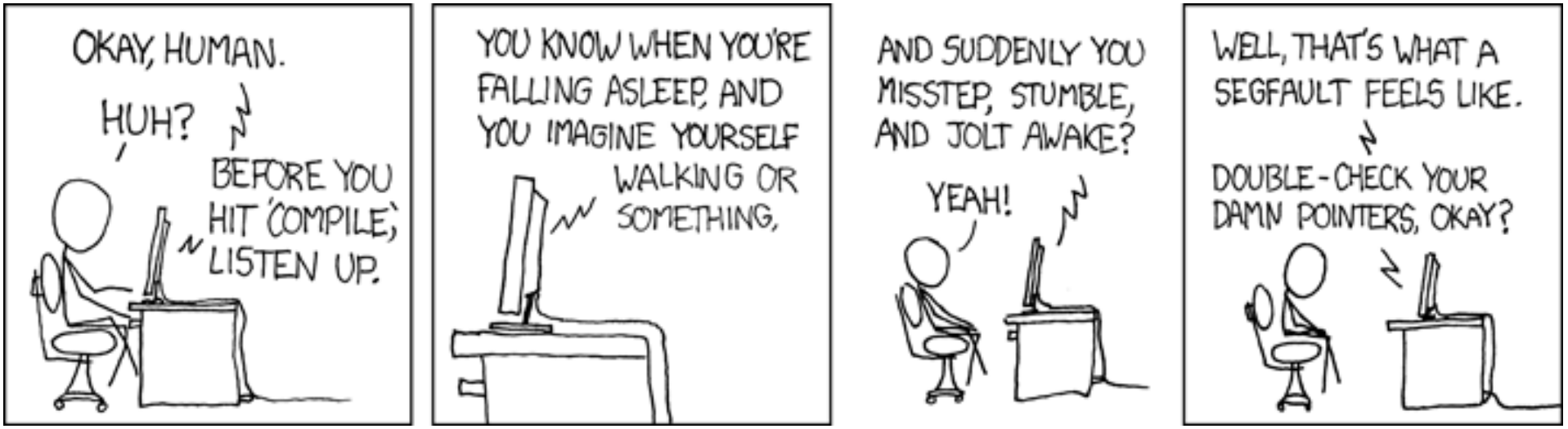
Pointers to Data Structures

```
typedef struct data {
    int n;
    char str[4];
} Data;
Data dat;
Data* ptr = &dat;
```



Segmentation Fault

- If you mess up your pointers (and you are lucky) you will get segmentation fault error during execution
- Caused by memory access violations when trying to access protected memory



Segmentation Fault

```
char str[5] = “Nope”;
char *s = str;
while(*s != '0')
  printf (“%s\n”, *s++);
```

Output??

Recursion

- *“To understand recursion, you must understand recursion”*
- Recursion in C programming – a function calling itself

```
void recursion() {
  recursion();
}
```

- Why?
  - Useful in writing concise programs
  - When a bigger problem can be broken down into smaller chunks of same problem

Recursion Example

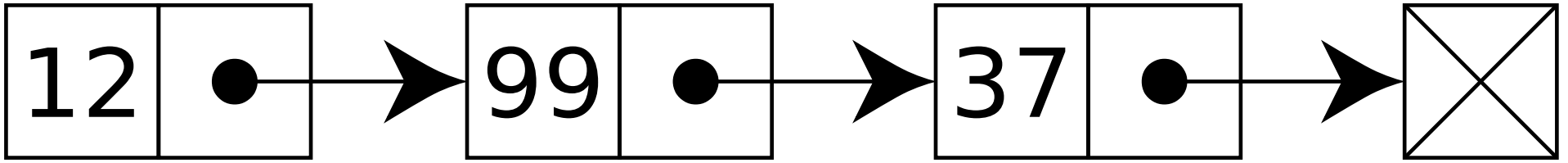
```
void main() {
    char str[5] = “Nope”;
    char *s = str;
    print(s);
}
```

```
void print(char *str) {
  if (*str == '\0')
    return;
  else {
    printf("%c\n", *str++);
    print(str);
  }
```

Linked List

- A sequence of data structures linked together by pointers pointing from one “node” to another
- Each node contains at least some data and a pointer to next(and/or previous) node in the list

```
struct node {
   int n;
   struct node * next;
};
```



Source: Lasindi

Linked List

- The pointer in the last item in the list is NULL
- Linked list terms:
  - Head: Pointer to the first node
  - Tail: Pointer to the last node
- Linked list operations:
  - Insertion
  - Deletion
  - Find
  - Sort

The Bash Demo

My machine setup:

- MacOS Sierra running
  - Oracle VirtualBox running
    - CentOS 6.3

Linked List Demo

Thank You!