# Structure & Union

Stanley Yao
Computer Science Department
University of Arizona

## Why we need structure?

- Each student: ID, name, major, year
- Data structure for each student is compound
  - `int ID;`
  - `char *name;`
  - `int major;`
  - `int year;`

BAD IDEAR

- 4 pieces of the student?



## Put Each Piece of People Together

```
struct student {
    int ID[30];
        char *name[30];
        int major[30];
        int year[30];
   };
```

- Like a catcard: A single card contains various information about you
- A *structure* is a collection of named data items. Each data item is called a *field* of the structure.



## Examples

```
struct date {
    char month;
       int day;
    int year;
   };
```

```
struct book {
    char *author;
       char *publisher;
       int year;
       int ISBN;
       float price;
   };
```



## Structure

- Structure declaration
- Structure variable declaration
  - `struct student s1, s2, s3;`
  - ```
    struct {
        int x;
        int y;
    } p1, p2, p3;
    ```
- Structure variable initialization
  - `struct date d = {6, 23, 2003};`



## Structure (cont.)

- Access the fields
  - "." operator
- Memory allocation and alignment
- sizeof()

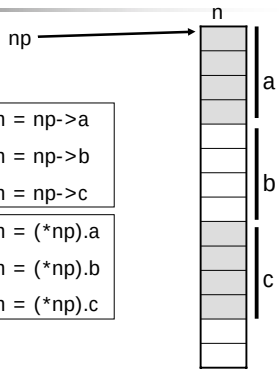
- Nested structures
- Recursive structures
    ```

## Structure Pointer

```
struct foo {
  int a;
  char b;
  int c;
} n, *np;
np = &n;
```

m = np->a
m = np->b
m = np->c

m = (*np).a
m = (*np).b
m = (*np).c






## Structure Operations

```
struct {
   int x;
   int y;
} a, b, *p;
```

- a = b;
- a.x = 2;
- p = &a;
- foo(a);
- return a;
- if (a == b) printf("equal"); // No!
  - Then how to compare?



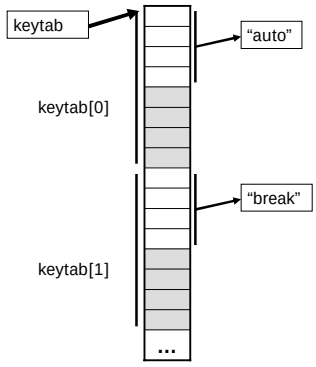
## Structure as Parameters

- Pass by value
  - Copy the whole structure
  - Less efficient
- Pass by reference
  - Only pass the pointer to the structure
  - More efficient in memory consumption



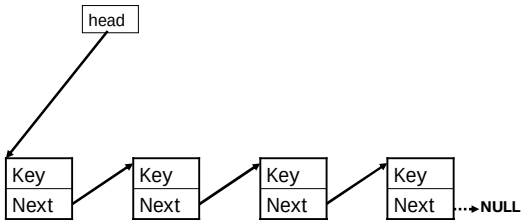
## Structure Array

```
struct key {
    char *word;
    int count;
} keytab[] = {
        {"auto", 0},
        {"break", 0},
        {"case", 0},
        {"char", 0},
        {"const", 0},
    };
```



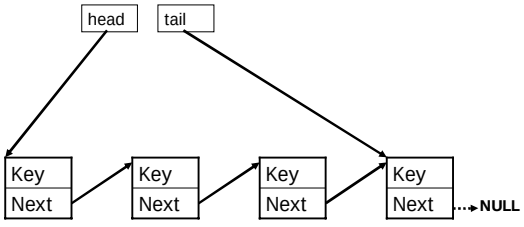



## Data Structures

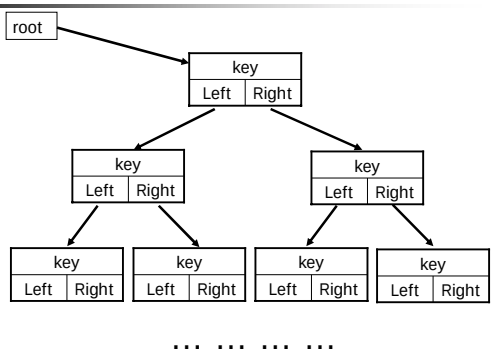





## Data Structures (cont.)



## Data Structures (cont.)






## typedef: Type Synonym

```
struct point {
   int x, y;
};
struct point p, *ptr;
```

```
typedef int WORD;
typedef char BYTE;
```

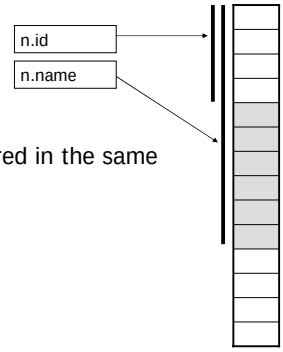
```
struct point {
   int x, y;
};
typedef struct point POINT;
POINT p, *ptr;
```

```
typedef struct {
   int x, y;
} POINT;
POINT p, *ptr;
```



## Union

```
union ident {
   int id;
   char name[10];
} n;
```




- All members are stored in the same memory space

```
struct student {
   int status;
   union ident id;
} n;
```



## Union (cont.)

- n.id = 1;
- strcpy(n.name, “Gates”);




## Acknowledgement


- John H. Hartman, *Classnotes for Csc352-Spring03*, CS Dept., University of Arizona, 2003
- Brian W. Kernighan, Dennis M. Ritchie, *The C Programming Language (2nd Ed.),* Prentice Hall, 1988