Linked Data Structures

- (singly) linked lists
- doubly linked lists
- Trees
In Java

```
Node
Text
T  p = null

Node L = new Node(t)
```

- \(\square\) = classes
- \(\bigcirc\) = objects
node

not data, but rather location of data

l.next = new Node(t2);

l-pointer

node

node

node

node
In C,
1) how to allocate memory
2) how do we bundle our "objects"
In array a

\[ a[ ] \]

\[ \ldots \]

\[ 2 \ 3 \ 6 \ 8 \ 1 \ 1 \ \ldots \]

\[ a \]

?\?

Items stored in successive locations

maxlors(?, )
We solve bundling with

```c
typedef struct {
    int a, j;
    node *next;
} node;
```

two fields in a node

definition of our node
node* l = (node*) malloc(sizeof(node));

node p;
node pj;

node* e;
`char* namej`

`name = getName();`

`char* getName();`
char *getName() {
    char temp[100];
    strcpy(temp, buffer);
    return temp;
}

name = getName();
Program will now **crash!!!**

never allocated space for name

temp has space allocated

but it's local (: not permanent)
Another problem

```c
char *name = malloc(100 * sizeof(char));
getNames(name);
free(name);
strcpy(name, 52);
printf("%s", name);
```

```c
void getNames(char *name) {
    // read data
    strcpy(name, buffer);
}
```
Yet another problem

char name[100];

for(int i=0; j<100; i++ )
{
    printf("*(name+i)\n");
}

array name = ptr

ptr graphical
float f[100];

(f+10) = ?

\{ f + 10 \times \text{sizeof}(float) \}
back to problem.....

char name[100];

char myc = *(name + 101);

(what happens?)

*(name + 101) = 'z';

may or may not crash

buffer overflow exploit
read User Data( & buffer );

LC

glensfully, happily
Stores all user data
Even though it's more dirty than
was asked for

→ if I'm clever

I can trick your program into running code of my choice.
In C

typedef struct {
    int a;
    node* next;
    node* prev;
} node;
typedef struct {
    int a;
    node* parent, left, right;
} node;
In Java
suppose \( p \) is a node object
w/ public field next

\[ \rightarrow \ p.\text{next} \]

In C, the syntax depends
on whether \( p \) is a pointer
struct node
{
    int data;
    node* next;
} node

node p;
p->next = node q;
(*q).next