UTS
UNIVERSITY OF TECHNOLOGY SYDNEY

POINTERS AND DYNAMIC MEMORY MANAGEMENT II

BEESHANGA ABEWARDANA JAYAWICKRAMA

UTS:
ENGINEERING AND
INFORMATION
TECHNOLOGY

UTS
UNIVERSITY OF TECHNOLOGY SYDNEY

DYNAMIC MEMORY MANAGEMENT

UTS:
ENGINEERING AND
INFORMATION
TECHNOLOGY

DYNAMIC ARRAYS USING POINTERS

Array size in C has to be a fixed value.

However, by dynamically allocating memory to a pointer, you can use pointers as dynamic length arrays.

Example shown in video lecture.

COMMON ERRORS – UNINITIALISED POINTERS

Like other data types, pointers must be initialised. Uninitialised pointers contain garbage addesses.

Good practice is to initialize to NULL.

```
char* cp;

cp = NULL;
```

Or allocate memory immediately.

```
char* cp;

cp = (char*) malloc(1);

/* I have malloc(1) instead of malloc(1*sizeof(char)).
Is that OK? Why? */
```

COMMON ERRORS – DANGLING POINTERS AND DOUBLE ALLOCATION

After deallocating memory you should manually point the pointer to NULL. Otherwise you could end up with

1) Dangling pointers – pointer points to a memory location not allocated for its use

2) Double deallocation – free memory after already having called free on the same block of memory

COMMON ERRORS – MEMORY LEAKS

Memory leaks are hard to find in a small program (unless you look at the code – in that case quite straightforward).

A symptom of improper memory management which causes major issues in large programs.

e.g.

```
char* cp;
cp = (char*) malloc(10);
cp = (char*) malloc(2);
```

X

```
char* cp;
cp =(char*) malloc(10);
free(cp);
cp =(char*) malloc(2);
```

UTS
UNIVERSITY OF TECHNOLOGY SYDNEY

POINTERS AND DYNAMIC MEMORY MANAGEMENT II

BEESHANGA ABEWARDANA JAYAWICKRAMA

UTS:
ENGINEERING AND
INFORMATION
TECHNOLOGY

UTS
UNIVERSITY OF TECHNOLOGY SYDNEY

LINKED LISTS

UTS:
ENGINEERING AND
INFORMATION
TECHNOLOGY

A POINTER CAN POINT TO ANYTHING: LINKED LISTS

Previous examples showed pointers to C primitive data types (e.g. int, char).

Pointer can point to a structure if we wanted to.

A structure packs a collection of data types.

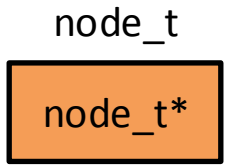
What if a pointer points to a structure, and that structure contains another pointer to a structure?

Such a data structure is called a Linked List.

LINKED LISTS

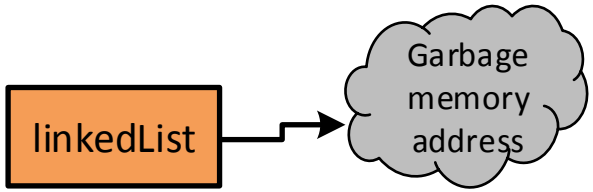
Consider the following structure:

```
struct node{
    struct node* nextp;
};
typedef struct node node_t;
```

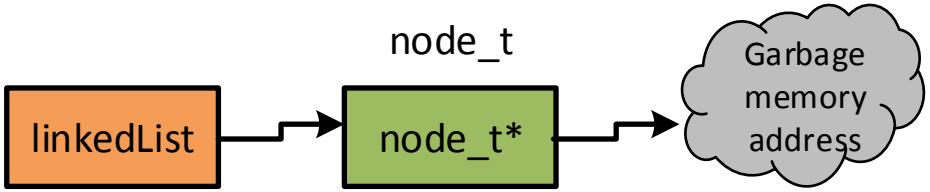


Consider the following memory allocation:

```
node_t* linkedListp;
```




```
linkedListp = (node_t*) malloc(sizeof(node_t));
```



LINKED LISTS

Access the `struct node*` pointed by `linkedListp`, we can write:

```
*linkedListp
```

That structure has one field called `next`. That can be accessed as:

```
(*linkedListp).next
```

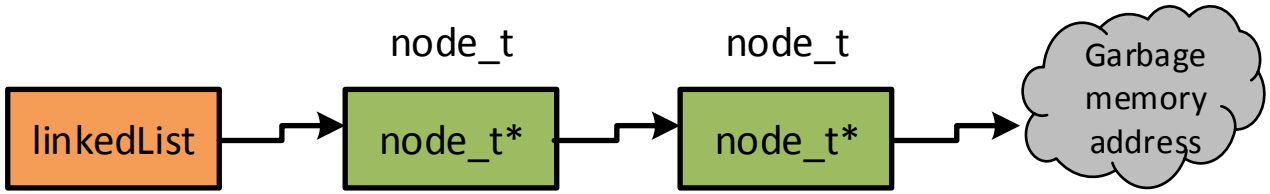
-or- `linkedListp -> next`

`(* ).` and `->` notations are identical

LINKED LISTS

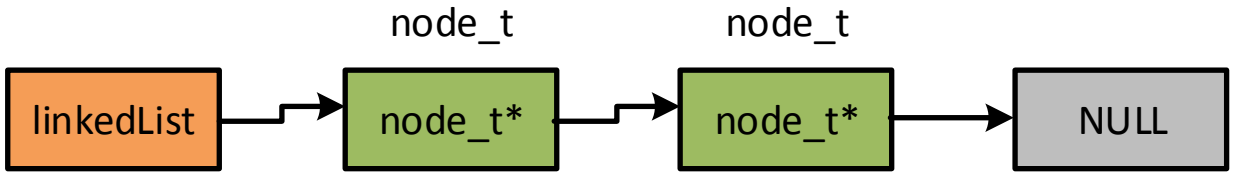
Because next is another pointer, we can allocate memory for it:

```
linkedListp -> nextp = (node_t*) malloc(sizeof(node_t));
```



This chain can be continued infinitely (until you run out of memory). By convention the end of a list is marked by a NULL pointer.

```
linkedListp -> nextp -> nextp = NULL;
```



TRAVELLING THROUGH LINKED LISTS

We can access any element in the linked list using the arrow (->) notation. For example we did:

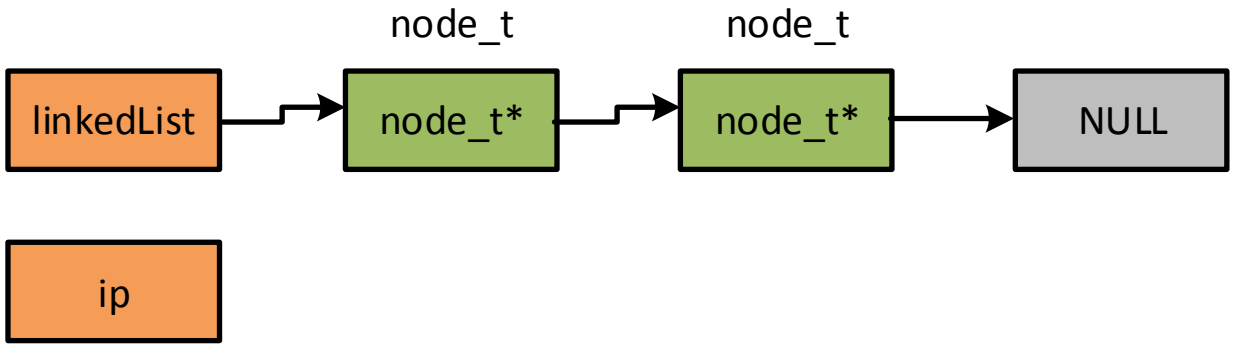
```
linkedListp -> nextp -> nextp = NULL;
```

But how can we travel through a linked list? Can we use the above method of appending arrow terms?

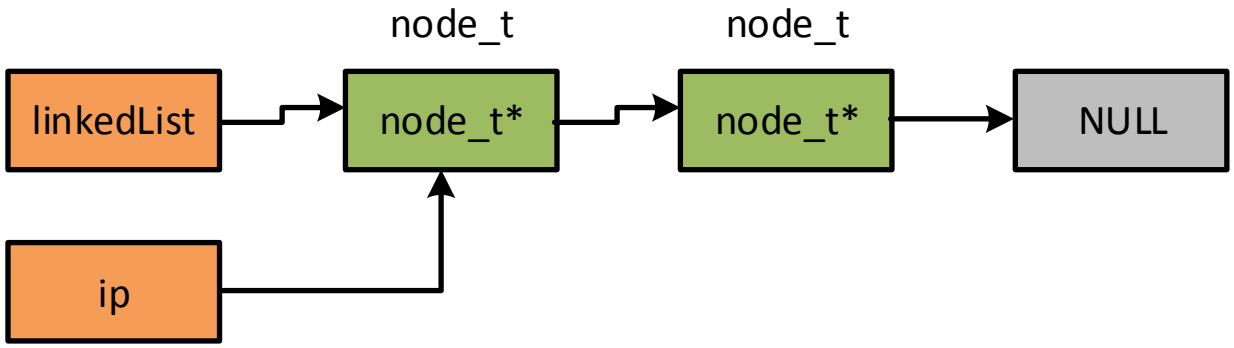
Remember in arrays we could use an index. There are no indices in linked lists (size is NOT fixed).

TRAVELLING THROUGH LINKED LISTS

Define a temporary pointer equivalent to an index. Let's call it `ip`

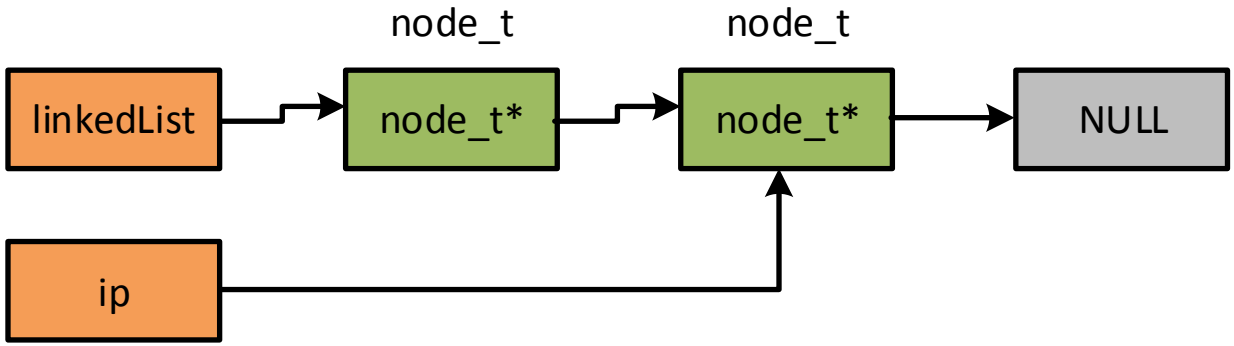


First node can be accessed as: `ip = linkedListp;`



TRAVELLING THROUGH LINKED LISTS

Similar to incrementing an index when accessing an array (i++) , next node in a linked list can be accessed as: `ip = ip -> nextp;`

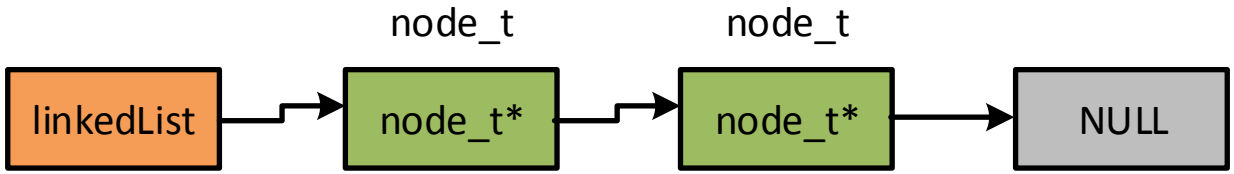


Food for thought: when should `ip` stop travelling?

AN EXAMPLE OF A MEMORY LEAK

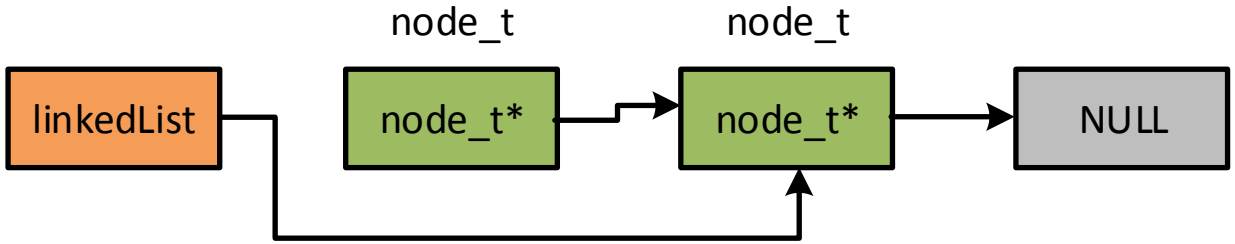
When doing dynamic memory management, we have to be really careful not to cause memory leaks.

Think about travelling through a linked list problem.



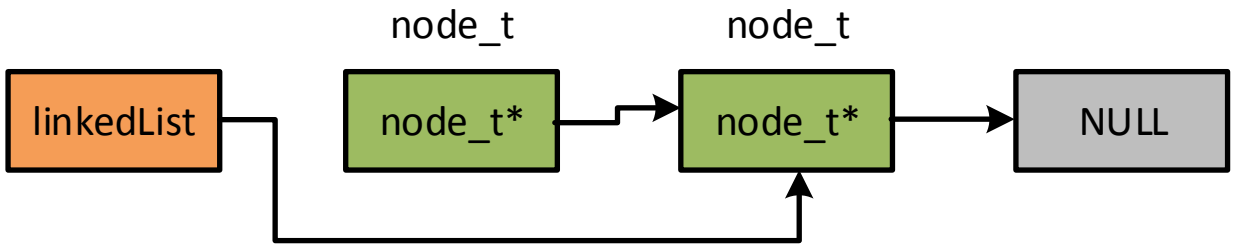
One can go to the second node by changing the `linkedListp` variable.

```
linkedListp = linkedListp -> nextp
```



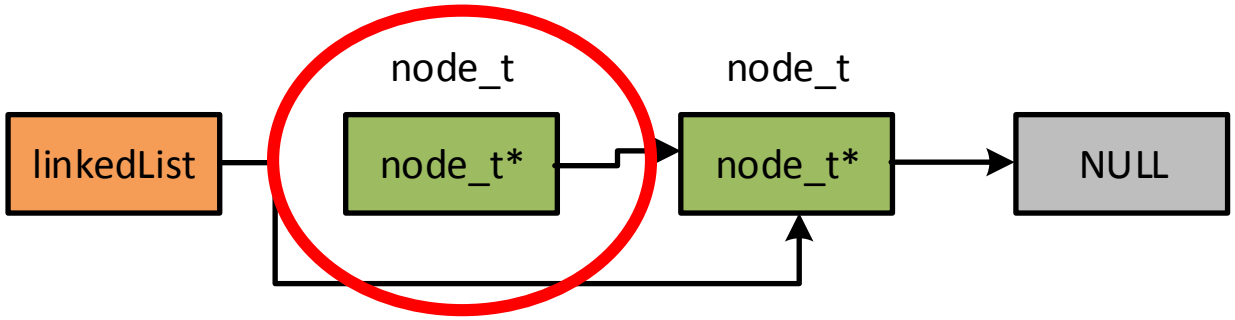
AN EXAMPLE OF A MEMORY LEAK

DO YOU SEE THE PROBLEM?



AN EXAMPLE OF A MEMORY LEAK

DO YOU SEE THE PROBLEM?



There is no way to access the first node again! Memory has been allocated for the program, but the program cannot make any use of that allocated memory.

If continued to travel along the list this way, you will lose all the memory allocated for the list.