

# ELEC2041

## Microprocessors and Interfacing

### Lecture 4: C-Language Review - III

<http://webct.edtec.unsw.edu.au/>

March, 2006

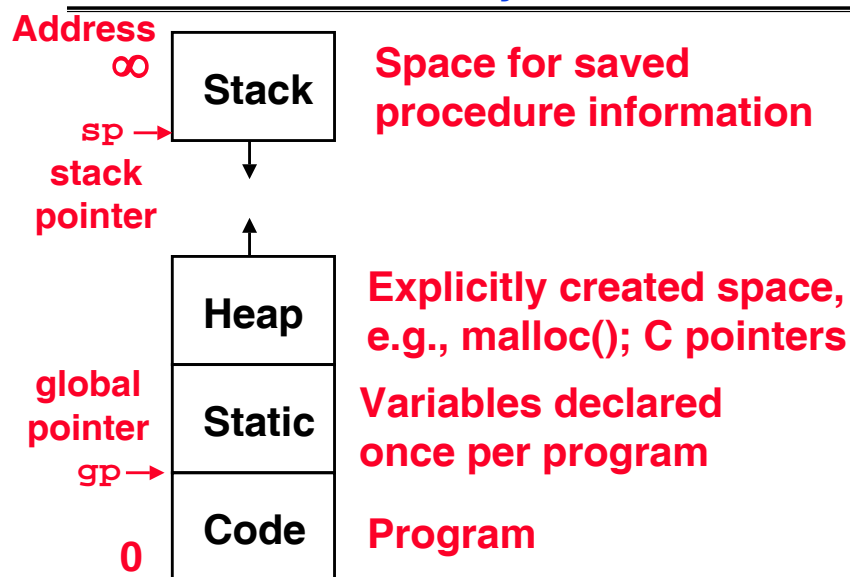
Saeid Nooshabadi

saeid@unsw.edu.au

## Overview

- Pointers to Pointers
- Linked List
- Shift Operators

## Review: C memory allocation



## Pointer Arithmetic Summary

- $x = *(p+1) \rightarrow x = *(p+1) ;$
- $x = *p+1 \rightarrow x = (*p) + 1 ;$
- $x = (*p)++ \rightarrow x = *p ; *p = *p + 1 ;$
- $x = *p++ = (*p++) = *(p)++ = *(p++) \rightarrow x = *p ; p = p + 1 ;$
- $x = ++p \rightarrow p = p + 1 ; x = *p ;$
- Lesson?
  - Avoid the confusing syntaxes!

## C String Standard Functions

- `int strlen(char *string);`
  - compute the length of `string`
- `int strcmp(char *str1, char *str2);`
  - return 0 if `str1` and `str2` are identical (how is this different from `str1 == str2`?)
- `char *strcpy(char *dst, char *src);`
  - copy the contents of string `src` to the memory at `dst`. The caller must ensure that `dst` has enough memory to hold the data to be copied.

## Pointers to pointers (#1/5)

- Sometimes you want to have a procedure increment a variable?
- What gets printed?

```
void AddOne(int x)
{
    x = x + 1;
}

int y = 5;
AddOne(y);
printf("y = %d\n", y);
```

**y = 5**

## Pointers to pointers (#2/5)

- Solved by passing in a pointer to our subroutine.
- Now what gets printed?

```
void AddOne(int *p)
{
    *p = *p + 1;
}

int y = 5;
AddOne(&y);
printf("y = %d\n", y);
```

**y = 6**

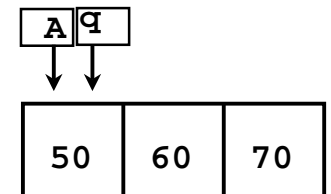
## Pointers to pointers (#3/5)

- But what if what you want changed is a pointer?
- What gets printed?

```
void IncrementPtr(int *p)
{
    p = p + 1;
}

int A[3] = {50, 60, 70};
int *q = A;
IncrementPtr(q);
printf("*q = %d\n", *q);
```

**\*q = 50**



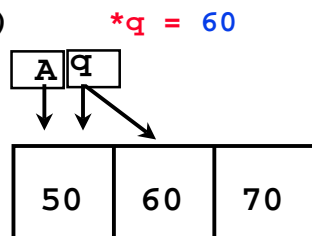
## Pointers to pointers (#4/5)

◦ **Solution! Pass a pointer to a pointer, called a handle, declared as `**h`**

◦ **Now what gets printed?**

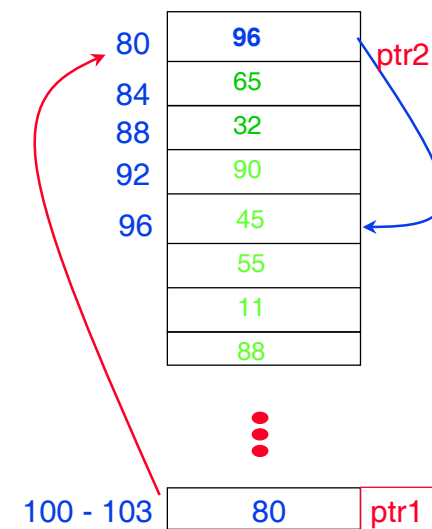
```
void IncrementPtr(int **h)
{   *h = *h + 1; }
```

```
int A[3] = {50, 60, 70};
int *q = A;
IncrementPtr(&q);
printf("*q = %d\n", *q);
```



## Pointers to pointers (#5/5)

```
int **ptr1
int *ptr2
```



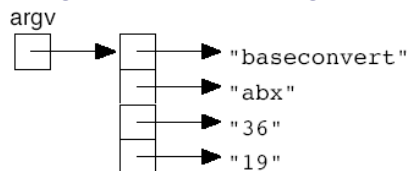
## Recall: C Syntax; Arguments to `main`

◦ **To get the main function to accept arguments, use this:**

```
int main (int argc, char *argv[])
```

◦ **What does this mean?**

- `argc` will contain the number of strings on the command line (the executable counts as one, plus one for each argument).
- `argv` is a pointer to an array containing the rest of the arguments as strings



## C structures : Overview

◦ **A `struct` is a data structure composed for simpler data types.**

- Like a class in Java/C++ but without methods or inheritance.

```
struct point {
    int x;
    int y;
};
void PrintPoint(point p)
{
    printf("(%d,%d)", p.x, p.y);
}
```

## C structures: Pointers to them

- The C arrow operator (->) dereferences and extracts a structure field with a single operator.
- The following are equivalent:

```
struct point *p;

printf("x is %d\n", (*p).x);
printf("x is %d\n", p->x);
```

## How big are structs?

- Recall C operator `sizeof()` which gives size in bytes (of type or variable)
- How big is `sizeof(p)`?

```
struct p {
    char x;
    int y;
};
```

- 5 bytes? 8 bytes?
- Compiler may word align integer y

## Peer Instruction

### Which are guaranteed to print out 5?

```
I: main() {
    int *a_ptr; *a_ptr = 5; printf("%d", *a_ptr);
}
```

```
II: main() {
    int *p, a = 5;
    p = &a; ...
    /* code; a & p NEVER on LHS of = */
    printf("%d", a); }
```

```
III: main() {
    int *ptr;
    ptr = (int *) malloc(sizeof(int));
    *ptr = 5;
    printf("%d", *ptr); }
```

	I	II	III
1:	-	-	-
2:	-	-	YES
3:	-	YES	-
4:	-	YES	YES
5:	YES	-	-
6:	YES	-	YES
7:	YES	YES	-
8:	YES	YES	YES

## Linked List Example (#1/7)

- Let's look at an example of using structures, pointers, `malloc()`, and `free()` to implement a linked list of strings.

```
struct Node {
    char *value;
    struct Node *next;
};
typedef Node *List;
```

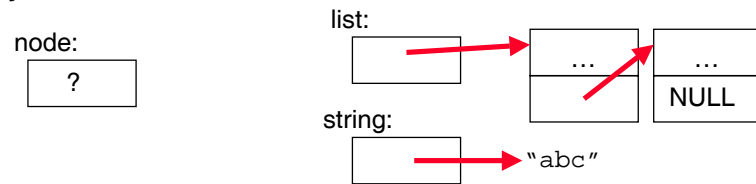
```
/* Create a new (empty) list */
List ListNew(void)
{ return NULL; }
```

## Linked List Example (#2/7)

```

/* add a string to an existing list */
List list_add(List list, char *string)
{
    struct Node *node =
        (struct Node*) malloc(sizeof(struct Node));
    node->value =
        (char*) malloc(strlen(string) + 1);
    strcpy(node->value, string);
    node->next = list;
    return node;
}

```



ELEC2041 lec04-C-language-III .17

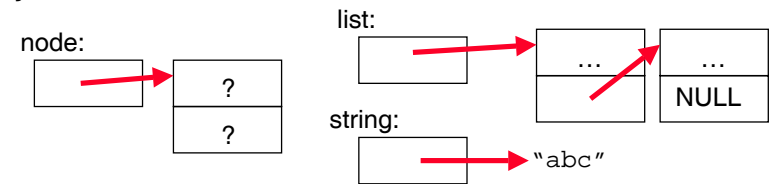
Saeid Nooshabadi

## Linked List Example (#3/7)

```

/* add a string to an existing list */
List list_add(List list, char *string)
{
    struct Node *node =
        (struct Node*) malloc(sizeof(struct Node));
    node->value =
        (char*) malloc(strlen(string) + 1);
    strcpy(node->value, string);
    node->next = list;
    return node;
}

```



ELEC2041 lec04-C-language-III .18

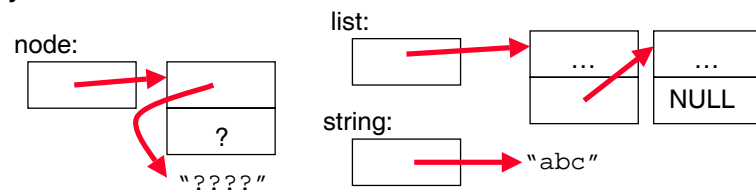
Saeid Nooshabadi

## Linked List Example (#4/7)

```

/* add a string to an existing list */
List list_add(List list, char *string)
{
    struct Node *node =
        (struct Node*) malloc(sizeof(struct Node));
    node->value =
        (char*) malloc(strlen(string) + 1);
    strcpy(node->value, string);
    node->next = list;
    return node;
}

```



ELEC2041 lec04-C-language-III .19

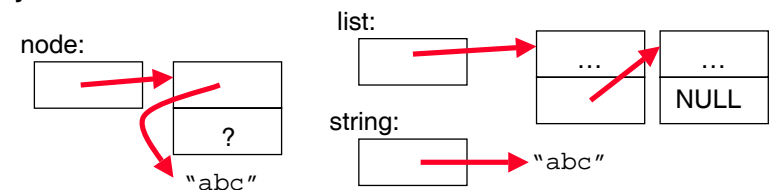
Saeid Nooshabadi

## Linked List Example (#5/7)

```

/* add a string to an existing list */
List list_add(List list, char *string)
{
    struct Node *node =
        (struct Node*) malloc(sizeof(struct Node));
    node->value =
        (char*) malloc(strlen(string) + 1);
    strcpy(node->value, string);
    node->next = list;
    return node;
}

```



ELEC2041 lec04-C-language-III .20

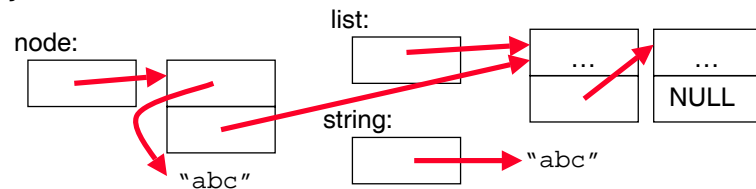
Saeid Nooshabadi

## Linked List Example (#6/7)

```

/* add a string to an existing list */
List list_add(List list, char *string)
{
    struct Node *node =
        (struct Node*) malloc(sizeof(struct Node));
    node->value =
        (char*) malloc(strlen(string) + 1);
    strcpy(node->value, string);
    node->next = list;
    return node;
}

```



ELEC2041 lec04-C-language-III .21

Saeid Nooshabadi

## Linked List Example (#7/7)

```

/* add a string to an existing list */
List list_add(List list, char *string)
{
    struct Node *node =
        (struct Node*) malloc(sizeof(struct Node));
    node->value =
        (char*) malloc(strlen(string) + 1);
    strcpy(node->value, string);
    node->next = list;
    return node;
}

```



ELEC2041 lec04-C-language-III .22

Saeid Nooshabadi

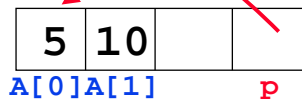
## Peer Instruction

```

int main(void){
    int A[] = {5,10};
    int *p = A;

    printf("%u %d %d %d\n", p, *p, A[0], A[1]);
    p = p + 1;
    printf("%u %d %d %d\n", p, *p, A[0], A[1]);
    *p = *p + 1;
    printf("%u %d %d %d\n", p, *p, A[0], A[1]);
}

```



If the first printf outputs 100 5 5 10, what will the other two printf output?

- 1: 101 10 5 10            then 101 11 5 11
- 2: 104 10 5 10            then 104 11 5 11
- 3: 101 <other> 5 10      then 101 <3-others>
- 4: 104 <other> 5 10      then 104 <3-others>
- 5: One of the two printf causes an ERROR
- 6: I surrender!

ELEC2041 lec04-C-language-III .23

Saeid Nooshabadi

## References:

- Nick Parlante: [Stanford CS Education Library](http://cslibrary.stanford.edu/) (<http://cslibrary.stanford.edu/>); A Collection of very useful material including:
- [Essential C: \(http://cslibrary.stanford.edu/101/\)](http://cslibrary.stanford.edu/101/) A relatively quick, 45 page discussion of most of the practical aspects of programming in C.
- [Binky Pointer Video: \(http://cslibrary.stanford.edu/104\)](http://cslibrary.stanford.edu/104/) Silly but memorable 3 minute animated video demonstrating the basic structure, techniques, and pitfalls of using pointers.
- [Pointer Basics: \(http://cslibrary.stanford.edu/106\)](http://cslibrary.stanford.edu/106/) The companion text for the Binky video.
- [Pointers and Memory: \(http://cslibrary.stanford.edu/102\)](http://cslibrary.stanford.edu/102/) A 31 page explanation of everything you ever wanted to know about pointers and memory.
- [Linked List Basics: \(http://cslibrary.stanford.edu/103\)](http://cslibrary.stanford.edu/103/) A 26 page introduction to the techniques and code for building linked lists in C.

ELEC2041 lec04-C-language-III .24

Saeid Nooshabadi

## Important Logical Operators

### Logical AND (&&) and bitwise AND (&) operators:

```
char a=4, b=8, c;
```

```
c = a && b;
```

```
/* After this statement  
c =1*/
```

```
c = a & b;
```

```
/* After this statement  
c =0*/
```

	Dec	Binary
a	4	0000 0100 > 0
b	8	0000 1000 > 0
a && b	True	
a & b	0	0000 0000

### Similarly logical OR (||) and bitwise OR (|) operators:

ELEC2041 lec04-C-language-III .25

Saeid Nooshabadi

## Important Shift Operators

### Logical shift left (<<) and shift right (>>) operators:

```
char a=4, b=8, c;
```

```
c = a << 2;
```

```
/* After this statement  
c =16*/
```

```
c = b >> 3;
```

```
/* After this statement  
c =1*/
```

	Dec	Binary
a	4	0000 0100
a << 2	16	0001 0000
b	8	0000 1000
b >> 3	1	0000 0001

ELEC2041 lec04-C-language-III .26

Saeid Nooshabadi

## What is this stuff good for?

- StressEraser: Biofeedback device promises to reduce stress
  - aid for deep breathing exercises, (prescribed to alleviate stress.)
  - The device tells when to inhale and when to stop.
  - It does this by divining the state of your nervous system by some clever analysis of your heart rate
- device is a pulse oximeter integrated with a display and a microprocessor.



- Pulse oximeters identify heartbeats by the variation in the amount of light absorbed through the skin of your finger as fresh blood pulses through it.
- It monitors heart rate to identify the activity level of the vagus nerve, one of 12 nerves that emanate directly from your brain rather than through your spinal cord.

ELEC2041 lec04-C-language-III .27

IEEE Spectrum March 2006

Saeid Nooshabadi

## World's Last C Bug

- If you remember nothing else, remember this:

```
while (1) {  
    status = GetRadarInfo();  
    if (status = 1) {  
        LaunchMissiles();  
    }  
}
```

= is used instead of ==

ELEC2041 lec04-C-language-III .28

Saeid Nooshabadi

## World's Last C Bug Improved!

```
launches = 0;
while (1){
    status = GetRadarInfo();
    if (status = 1){
        LaunchMissiles();
        launches++;
    }
    if (launches > 1){
        apologize();
    }
}
```

Steve Litt: [www.troubleshooters.com](http://www.troubleshooters.com)

## Example #1:

### How many bugs in this code?

```
#include <stdio.h>
int main ( ) {
    int numAs; /* counts A's in the input */
    char c;
    while (c = getchar ( ) != EOF) {
/* getchar returns EOF if no more chars to read. */
        if (c = 'A') {
            numAs++;
        }
    }
    printf ("%d A's in the input\n", numAs);
    return 0;
}
```

Choices:  
1 or none,  
2,  
3,  
4,  
5 or more

## Example #1 (Solution):

### How many bugs in this code?

```
#include <stdio.h>
int main ( ) {
    int numAs; /* counts A's in the input */
    char c;
    numAs = 0;
    while ((c = getchar ( )) != EOF) {
/* getchar returns EOF if no more chars to read. */
        if (c == 'A') {
            numAs++;
        }
    }
    printf ("%d A's in the input\n", numAs);
    return 0;
}
```

3 Bugs

## Bug Symptoms

- A value that's wildly out of range suggests an uninitialized variable or function return value.
- A loop that's executed 0 times or the maximum number of times when it shouldn't be suggests misuse of = in a test, or misparenthesization.

## Example #2

◦ What output is produced by this code?

```
void addOne (int x) {  
    x = x + 1;  
}  
  
int main ( ) {  
    int y = 3;  
    addOne (y);  
    printf ("%d, y);  
    return 0;  
}
```

Choices:  
3,  
4,  
unknown,

ELEC2041 lec04-C-language-III .33

Saeid Nooshabadi

## Example #2 (Solution)

◦ What output is produced by this code?

```
void addOne (int x) {  
    x = x + 1;  
}  
  
int main ( ) {  
    int y = 3;  
    addOne (y);  
    printf ("%d, y);  
    return 0;  
}
```

3 is Produced  
**Passed by value:**  
**Make a copy of the**  
**original argument.**  
**The original won't**  
**change**

ELEC2041 lec04-C-language-III .34

Saeid Nooshabadi

## Example #3

◦ What choice for the blank ensures that 7 = 7 is printed?

```
#include <stdio.h>  
int *f (int varNotToSet, int varToSet) {  
    int n = 7;  
    varToSet = 7;  
    return _____ ;  
    /* &n, &varNotToSet, or &varToSet could go here */  
}  
  
int main ( ) {  
    int *ptr;  
    int k1 = 7, k2 = 0;  
    ptr = f(k1, k2);  
    printf ("7 = ");  
    printf ("%d\n", *ptr);  
    return 0;  
}
```

Choices:  
1) &n, 2) &varNotToSet,  
3) &varToSet  
1  
2,  
3,  
none

ELEC2041 lec04-C-language-III .35

Saeid Nooshabadi

## Example #3 (Solution)

**Answer: none**

- Variables and function parameters are allocated on the system stack
- When a function exits, its allocated space gets reallocated to another function.

ELEC2041 lec04-C-language-III .36

Saeid Nooshabadi

## Example #4

◦ What choice for the blank ensures that values[0]=x

```
int main ( ){
    int values[20];
    int x;
    ...
    assign ( _____ , x);
    ...
}
```

Answer:

1. values
2. &values[0]

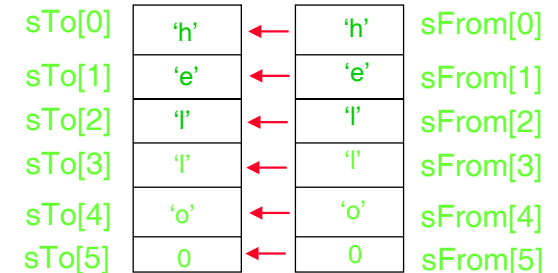
```
void assign ( int *y , int x) {
    *y = x;
}
```

## Example #5

◦ How to copy one array to another array

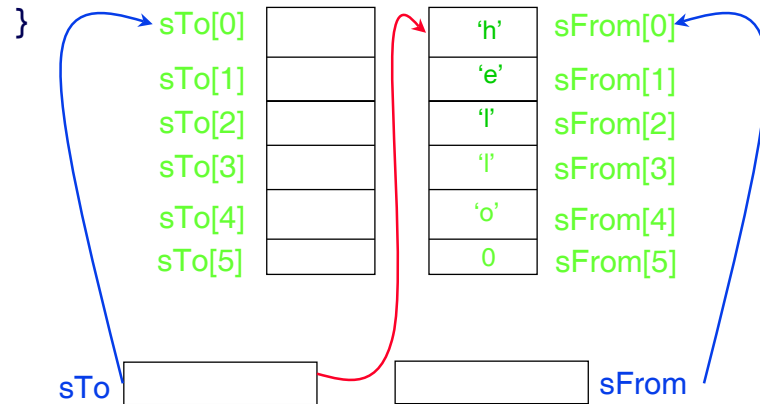
```
int main ( ){
    char sFrom[6], sTo[6];
    copy (sTo[6], sFrom[6]);
}

void copy (char sTo[], char sFrom[]) {
    ----
    ----
}
```



## Example #5 (Solution #1/4)

```
void copy (char sTo[], char sFrom[]) {
    sTo = sFrom;
}
```

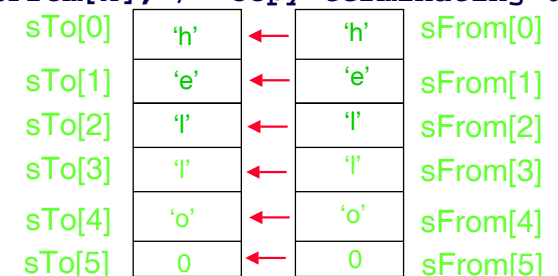


Similarly you don't compare two string using ==

## Example #5 (Solution #2/4)

◦ Straight Forward Array Version

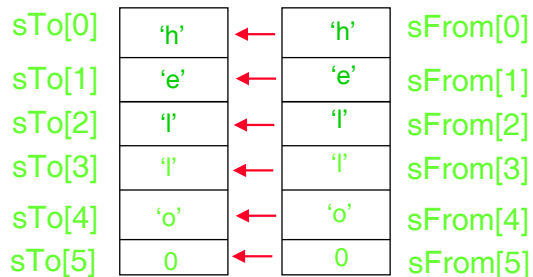
```
void copy (char sTo[], char sFrom[]) {
    int k = 0;
    while (sFrom[k] != '\0') {
        sTo[k] = sFrom[k];
        k++;
    }
    sTo[k] = sFrom[k]; /* copy terminating 0
*/
}
```



## Example #5 (Solution #3/4)

### ◦ Array Version (Taking advantage of value returned by assignment operator)

```
void copy (char sTo[], char sFrom[]) {
    int k = 0;
    while ((*sTo = sFrom[k]) != '\0'){
        k++;
    }
}
```



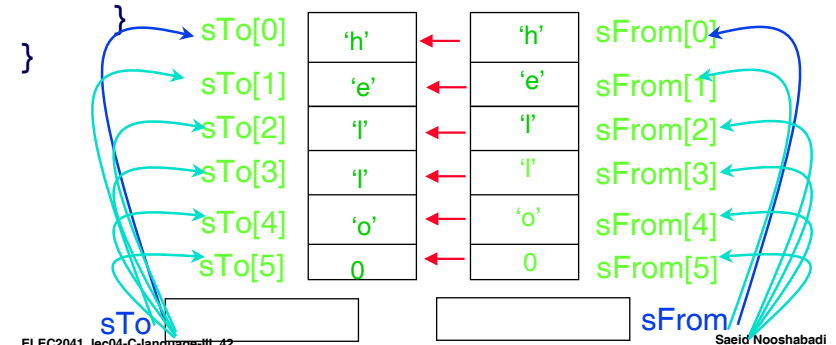
ELEC2041 lec04-C-language-III .41

Saeid Nooshabadi

## Example #5 (Solution #3/4)

### ◦ Pointer Version

```
void copy (char sTo[], char sFrom[]) {
    while ((*sTo = *sFrom) != '\0') {
        /* pointer arithmetic (K&R 5.4) */
        sFrom++;
        sTo++;
    }
}
```



ELEC2041 lec04-C-language-III .42

Saeid Nooshabadi

## Things to Remember (#1/2)

- Use handles to change pointers
- Create abstractions with structures
- A `struct` is a data structure composed for simpler data types.
- We can change the value of a pointer in a function by pointer to pointer feature.

ELEC2041 lec04-C-language-III .43

Saeid Nooshabadi

## Things to Remember (#2/2)

- Pointers and arrays are virtually same
- C knows how to increment pointers
- C is an efficient language, with little protection
  - Array bounds **not checked**
  - Variables **not** automatically initialized
- (Beware) The cost of efficiency is more overhead for the programmer.
  - “C gives you a lot of extra rope but be careful not to hang yourself with it!”

ELEC2041 lec04-C-language-III .44

Saeid Nooshabadi