Embedded Systems Interfacing

Lecture Ten

Dynamic Memory Allocation

Basic RAM Sections

Global and Static Variables

Local Variables, Function Arguments and return

Dynamic Allocated Data

Statically Allocated Data

Stack

Heap

RAM

Random
Access
Memory

Usually so far, the arrays must have *fixed* length (i.e., length is known at compile time).

```
/* Create Array of 11 Element */
char arr[11];
```

Length must be defined at compilation time, *it can't be a variable* ! For that, arrays are statically allocated. But what if we need to create an array with a size defined by the user in the run time … ?

This is *not doable* by the normal arrays, but it can be done by *Dynamic Allocation*.

\<stdlib.h\>

Function	Description
malloc()	Allocates requested size of bytes and returns a pointer first byte of allocated space
calloc()	Allocates space for an array elements, initializes to zero and then returns a pointer to memory
free()	De-allocate the previously allocated space
realloc()	Change the size of previously allocated space

malloc ( )

The function *malloc()* reserves a block of memory of specified size and return a ***pointer to void*** which can be ***casted*** into pointer of any form.

Syntax

pointer = (cast_type*)   malloc (Size_To_Reserve_In_Bytes)

Example

```
u8 *ptr;
ptr = (u8*) malloc(10);
```

Note:
Input to *malloc* function can be variable as it reserve the memory in the run time when the processor executes the instruction.

Write a C code that apply the bubble sorting algorithm on a set of numbers entered by the user. The code shall ask the user to enter the number of values to be sorted, then the code shall ask the user to enter the values then print them in ascending order.

```
Please Enter the number of numbers: 5
Please Enter number 1: 4
Please Enter number 2: 21
Please Enter number 3: 1
Please Enter number 4: 7
Please Enter number 5: 25
Values after sorting are:
1
4
7
21
25
```

Time To Code

calloc ( )

The only difference between *malloc()* and *calloc()* is that *malloc()* allocates single block of memory whereas *calloc()* allocates **multiple blocks** of memory each of **same size** and sets **all bytes to zero**. The name *calloc* stands for "*contiguous allocation*".

Syntax

pointer = (cast_type*) malloc (*number_of_Blocks_To_Reserve* , *Size_Of_Block_In_Bytes*)

Example

```
u32 *ptr;
ptr = (u32*) calloc(10,sizeof(u32));
```

This code reserve an array of 10 elements of type u32

malloc vs calloc

malloc()	calloc()
Stands for *memory allocation*	Stands for *contiguous allocation*
takes one argument that is, ***number of bytes***	take two arguments those are: ***number of blocks*** and ***size of each block***
returns a pointer to n bytes of *uninitialized storage*	returns a pointer to enough free space for an array of n objects of the specified size *initialized to zero*
malloc is *faster* than *calloc*. *calloc* takes little longer than *malloc* because of the extra step of initializing the allocated memory by zero	

Dynamically allocated memory created with either *calloc()* or *malloc()* **doesn't get freed** on its own even if it is defined locally in a function. You must explicitly use free() to release the space.

free()

This function frees the space allocated in the memory pointed by **pointer**.

<u>Syntax</u>

free (*pointer_to_block_to_free*)

<u>Example</u>

```c
u32 *ptr;
ptr = (u32*) calloc(10,sizeof(u32));

/*
    Some Code
*/

free(ptr)
```

realloc( )

If the previously allocated memory is *insufficient* or more than required, you can change the previously allocated memory size using realloc().

Syntax

realloc(*pointer_to_previously_allocated_block* **,** *new_size***)**

Example

```
u32 *ptr;
/* Allocate 10 u32 variables */
ptr = (u32*) malloc(10*size(u32));

/*
    Some Code
*/

/* Expand the reserved area to 20 u32 variables*/
realloc(ptr,20*size(32))
```
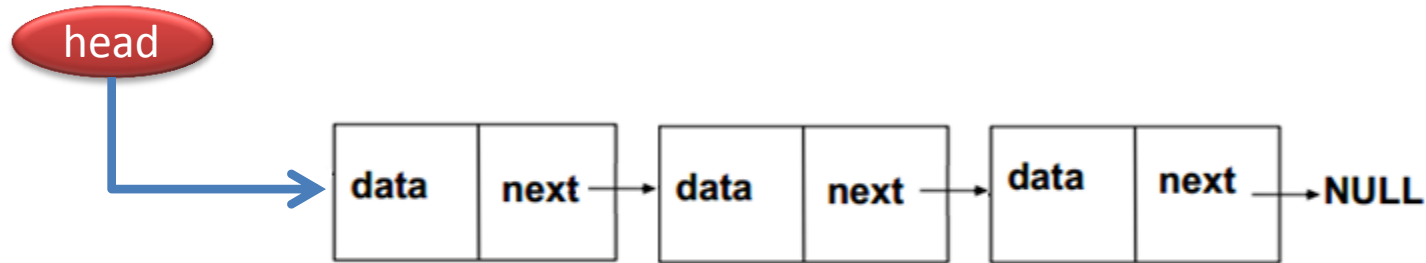
Linked List

A linked list is a type of data structure that contains a *list of data* of any type like numbers, characters or any other type. It looks like an array but the main *difference* is that the linked list length can be *changed dynamically*. **i.e.** It can be increased or decreased in the run time.

Each element of the list must have a *link to next element*. The last element in the list shall be linked to NULL.

The linked list is consisted of some elements each one called a *node*. The node mainly consists of a *data* and a *pointer* to the next node. Keep in mind that each node must points to the next node to keep the list connected.
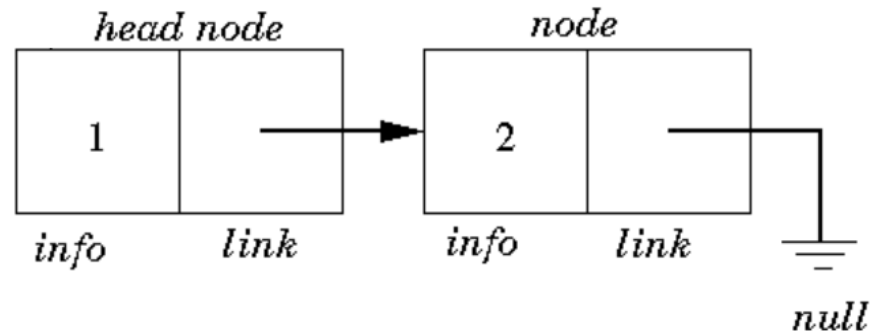

The node simply is a structure has two elements, a data type and a pointer. ***For example***:

```c
typedef struct Node_type node;

struct Node_type{
    u32 value;
    node *Next;
};
```
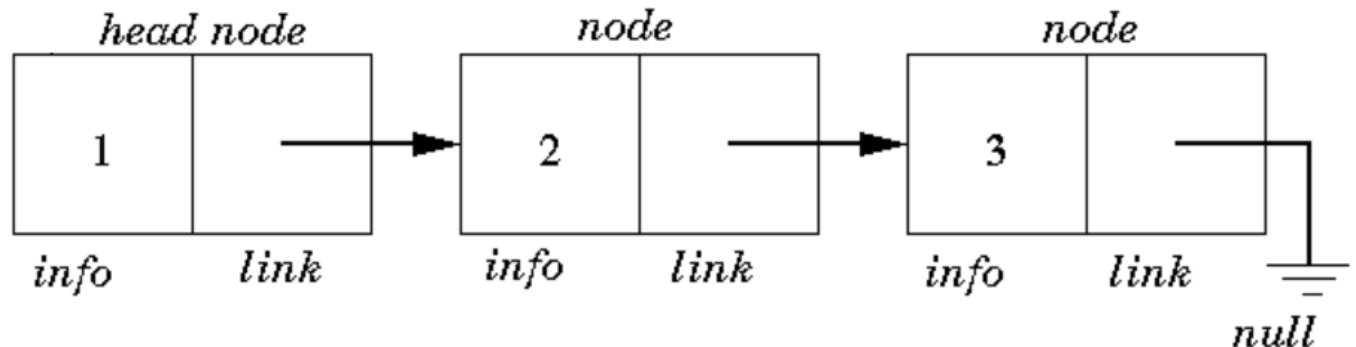
To add new element to the linked list, you shall create a node dynamically. Assign the data to the node. Then connect the next pointer of the last created node to the new node.

List before adding Node 3

List after adding Node 3

Write a C code that allow the user to add a node to linked list by pressing 0 and to print the linked list by pressing 1. The software shall continue forever till the user enters 2. If the user enters any other numbers the software shall print incorrect entry. See the following scenarios for more details.

Scenario 1 Add Node

```
To add node enter 0
To print the linked list enter 1
To exit press to 2
Your Choice: 0
Please Enter Node Value: 10
Node Added Thank You
```

Time To Code

Scenario 2 Print Linked List

Write a C code that allow the user to add a node to linked list by pressing 0 and to print the linked list by pressing 1. The software shall continue forever till the user enters 2. If the user enters any other numbers the software shall print incorrect entry. See the following scenarios for more details.

```
To add node enter 0
To print the linked list enter 1
To exit press to 2
Your Choice: 1


----------------------
Node Number 1 = 10
Node Number 2 = 20
Node Number 3 = 30
----------------------
```

Time To Code

Scenario 3 Print empty List

Write a C code that allow the user to add a node to linked list by pressing 0 and to print the linked list by pressing 1. The software shall continue forever till the user enters 2. If the user enters any other numbers the software shall print incorrect entry. See the following scenarios for more details.

```
To add node enter 0
To print the linked list enter 1
To exit press to 2
Your Choice: 1

---------------------
List is Empty
---------------------
```

Time To Code

IMT
S C H O O L

Write a C code that allow the user to add a node to linked list by pressing 0 and to print the linked list by pressing 1. The software shall continue forever till the user enters 2. If the user enters any other numbers the software shall print incorrect entry. See the following scenarios for more details.

Scenario 4 Wrong Entry

```
To add node enter 0
To print the linked list enter 1
To exit press to 2
Your Choice: 5
Invalid Choice please try again
```

Time To
Code

Scenario 5 Exit

Write a C code that allow the user to add a node to linked list by pressing 0 and to print the linked list by pressing 1. The software shall continue forever till the user enters 2. If the user enters any other numbers the software shall print incorrect entry. See the following scenarios for more details.

```
To add node enter 0
To print the linked list enter 1
To exit press to 2
Your Choice: 2
Thank You
Good Bye
```

Time To

Code

Complete lab 2 by adding a new function that allow the user to delete certain node. The user shall enter a value then the code shall delete any node that holds this value

Complete lab 2 by adding a new function that allows the user to add a new node at a certain position. The code shall ask the user to enter a value of the node and the node index. the code shall add the new node to the received index and shift all nodes to right.
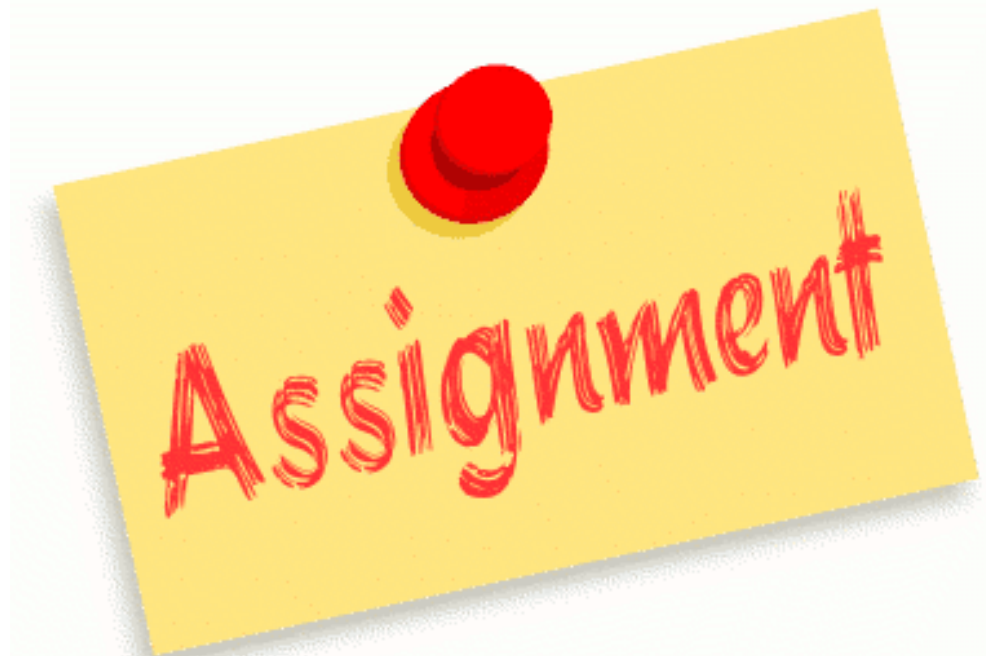
www.imtschool.com

ww.facebook.com/imaketechnologyschool/