OTHER SUPPLEMENTAL NOTES

Intro to Dynamic Memory Allocation & Linked Lists in C

```
(int *) malloc(sizeof(int))
```

type cast to change to pointer of desired type

function: allocates the memory (and returns address of memory allocated as pointer to type void)

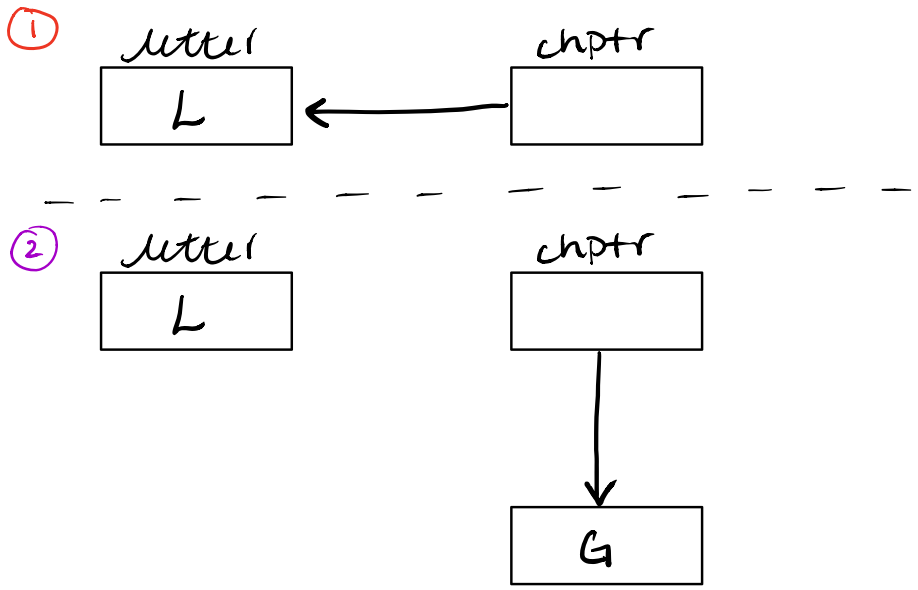
sizeof(): function, in this case allocates enough bytes to store an integer

this will allocate enough space for an integer and casts the address of the location to an int pointer

```
intptr = (int *) malloc(sizeof(int));
```


```
*intptr = 30;
```

```c
// Example 1


#include <stdio.h>
#include <stdlib.h>


int main()
{
    char letter = 'L',
         *chptr;

    chptr = &letter;
    printf("letter is %c\n", letter);
    printf("*chptr is %c\n", *chptr);

    chptr = (char *) malloc(sizeof(char));
    *chptr = 'G';
    printf("letter is %c\n", letter);
    printf("*chptr is %c\n", *chptr);

    free(chptr);
    return 0;
}
```

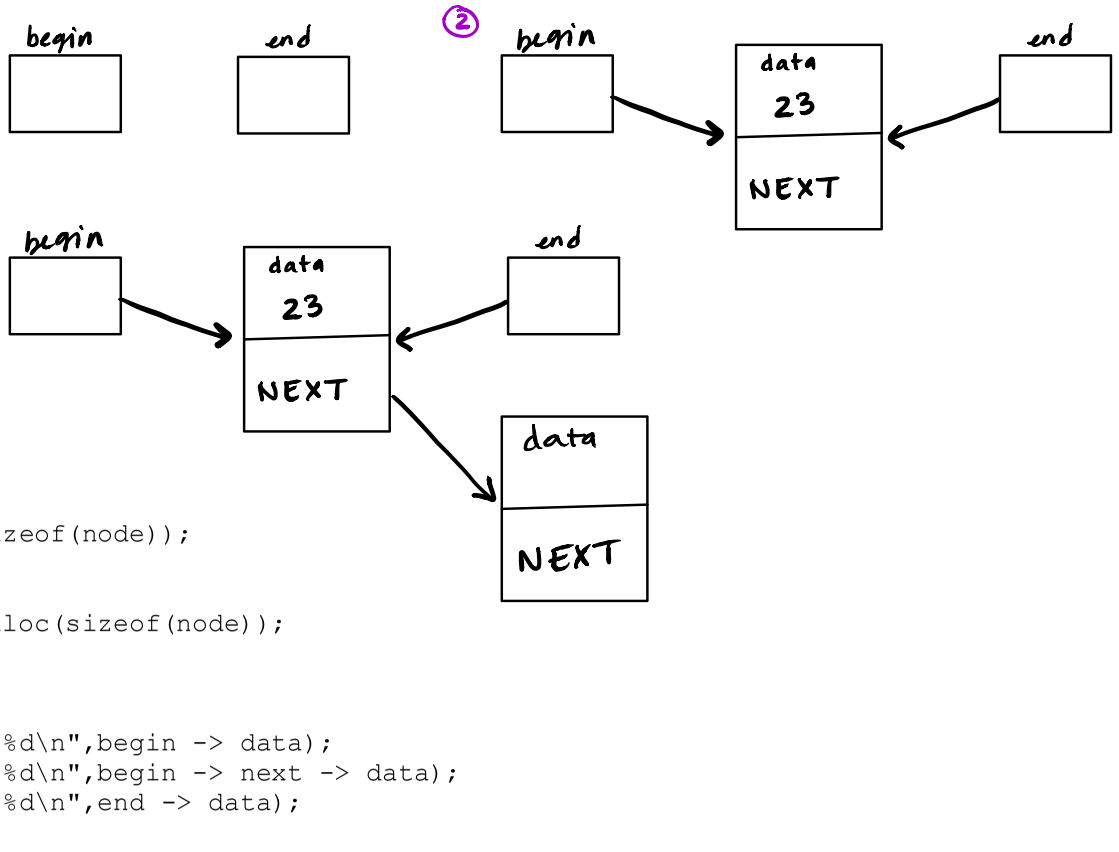


output:

letter is L
*chptr is L
letter is L
*chptr is G

// Example 2 (from GPP C Ch.5 AB, problem # 4a)

```
#include <stdio.h>
#include <stdlib.h>


typedef struct linked
{
    int data;
    struct linked *next;
}  node;
typedef node * nodeptr;


int main()
{
    nodeptr begin,
            end;
    begin = (nodeptr) malloc(sizeof(node));
    end = begin;
    end -> data = 23;
    end -> next = (nodeptr) malloc(sizeof(node));
    end = end -> next;
    end -> data = 11;
    end -> next = NULL;
    printf("The data member is %d\n",begin -> data);
    printf("The data member is %d\n",begin -> next -> data);
    printf("The data member is %d\n",end -> data);
    free(begin);
    free(end);
    return 0;
}
```



Output:

The data member is 23
The data member is 11
The data member is 11