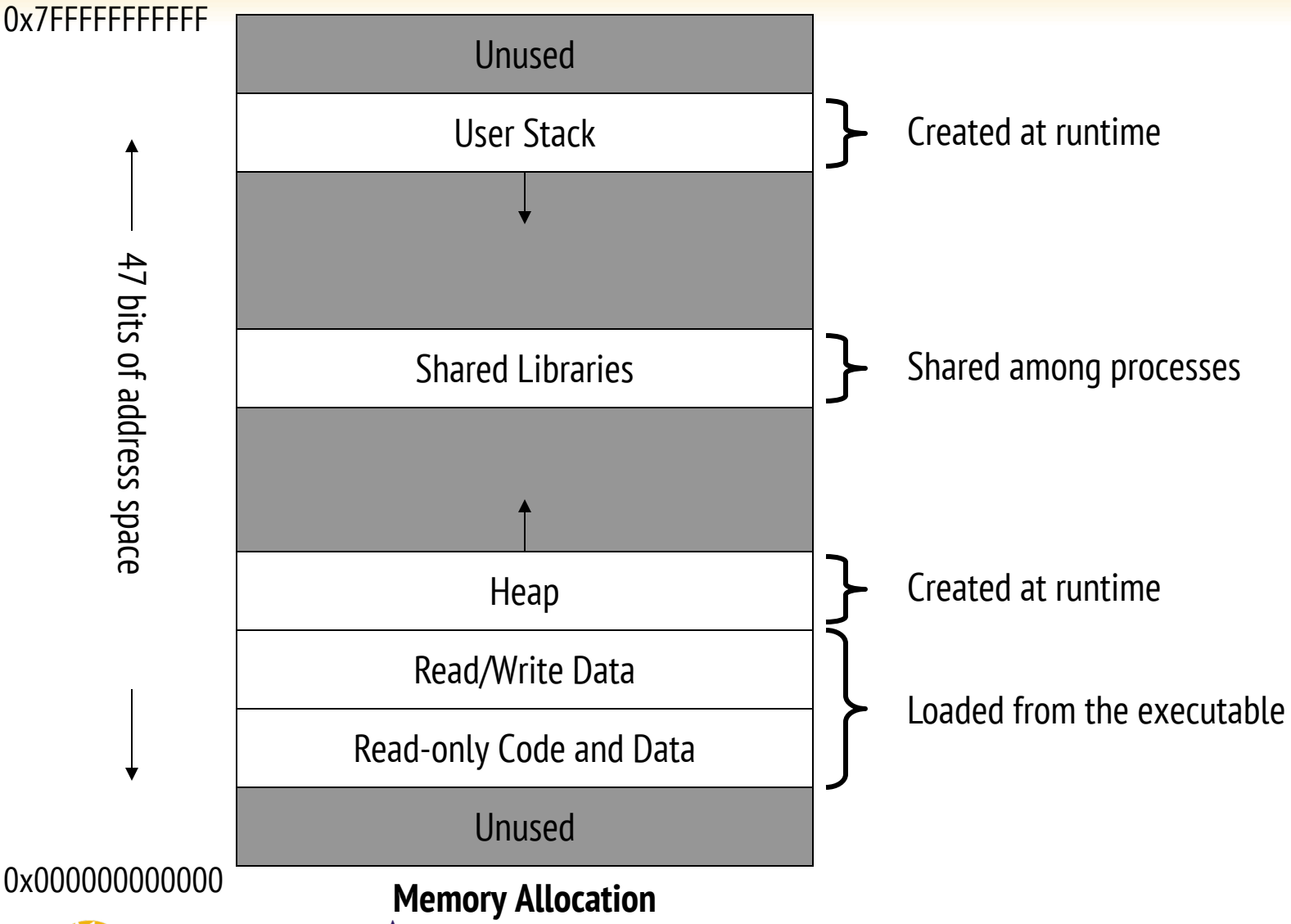
Programming for Engineers
Dynamic Memory Allocation

ICEN 200 – Spring 2018

Prof. Dola Saha

A Running Program's Memory



Memory Allocation

Allocation

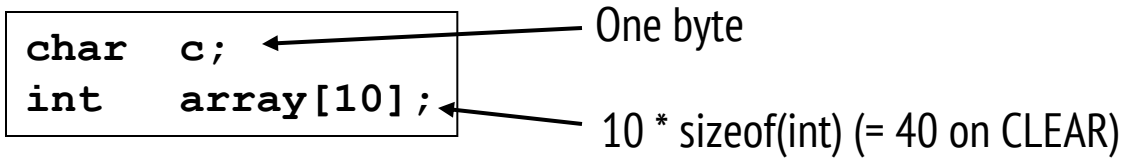
- For all data, memory must be *allocated*
  - Allocated = memory space reserved

- Two questions:
  - When do we know the size to allocate?
  - When do we allocate?

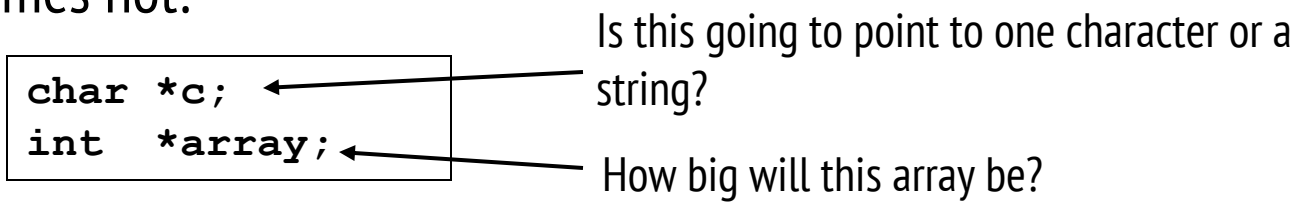
- Two possible answers for each:
  - Compile-time (*static*)
  - Run-time (*dynamic*)

How much memory to allocate?

- Sometimes obvious:



- Sometimes not:



  - How will these be used???
    - Will they point to already allocated memory (what we've seen so far)?
    - Will new memory need to be allocated (we haven't seen this yet)?

Dynamic Memory Allocation

- Creating and maintaining dynamic data structures requires dynamic memory allocation—the ability for a program to *obtain more memory space at execution time* to hold new nodes, and to *release space no longer needed*.
- Functions malloc and free, and operator sizeof, are essential to dynamic memory allocation.

Memory Use

- heap
  - region of memory in which function malloc dynamically allocates blocks of storage
- stack
  - region of memory in which function data areas are allocated and reclaimed

malloc()

- *void* * **malloc** *(size_t size)*
- Input: number of bytes to be allocated
- Output: a pointer of type void * (pointer to void) to the allocated memory.
- A void * pointer may be assigned to a variable of *any* pointer type.
- Example:

```
newPtr = malloc(sizeof(int));
```

- The allocated memory is *not* initialized.
- If no memory is available, malloc returns NULL.

free()

- Function free *deallocates* memory—i.e., the memory is *returned* to the system so that it can be reallocated in the future.
- To *free* memory dynamically allocated by the preceding malloc call, use the statement
  - `free(newPtr);`
- C also provides functions calloc and realloc for creating and modifying *dynamic arrays*.

calloc() and realloc()

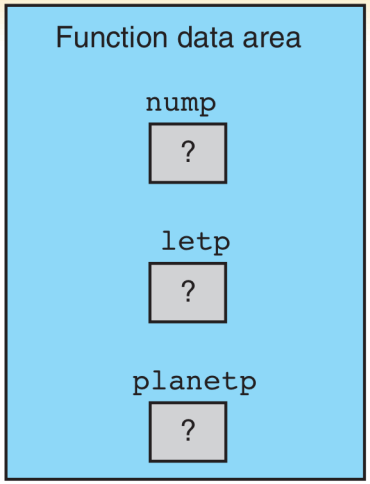
- `calloc()`
  - Allocates memory and cleares it to 0.
  - *`void * `***`calloc`***` (size_t count, size_t eltsize)`*
- `realloc()`
  - Make a block previously allocated by malloc larger or smaller, possibly by copying it to a new location.
  - `void *realloc (void *`*`addr`*`, size_t `*`size`*`)`

Dynamic Memory Allocation

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

int main()
{
   char *str;

   /* Dynamic Memory allocation */
   str = (char *) malloc(50);
   if (str == NULL)
   {
       printf("Error in memory allocation.");
       return(1);
   }

   strcpy(str, "Programming is fun!");
   printf("String = %s\n", str);

   // Free the memory allocated
   free(str);

   return(0);
}
```

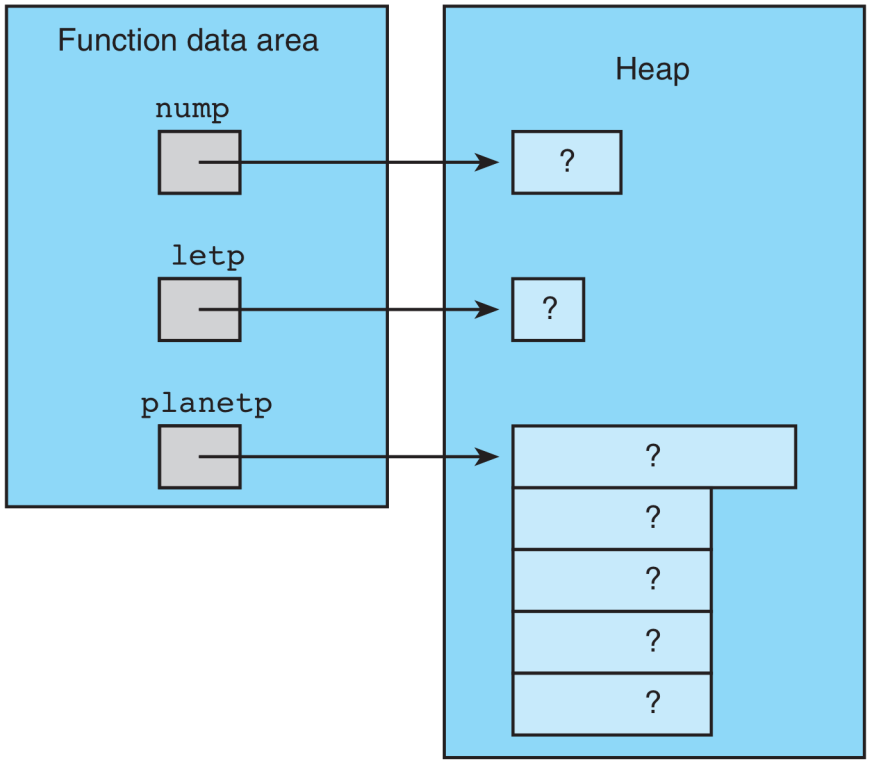
Memory Allocation (1)

```
int       *nump;
char      *letp;
planet_t  *planetp;
```



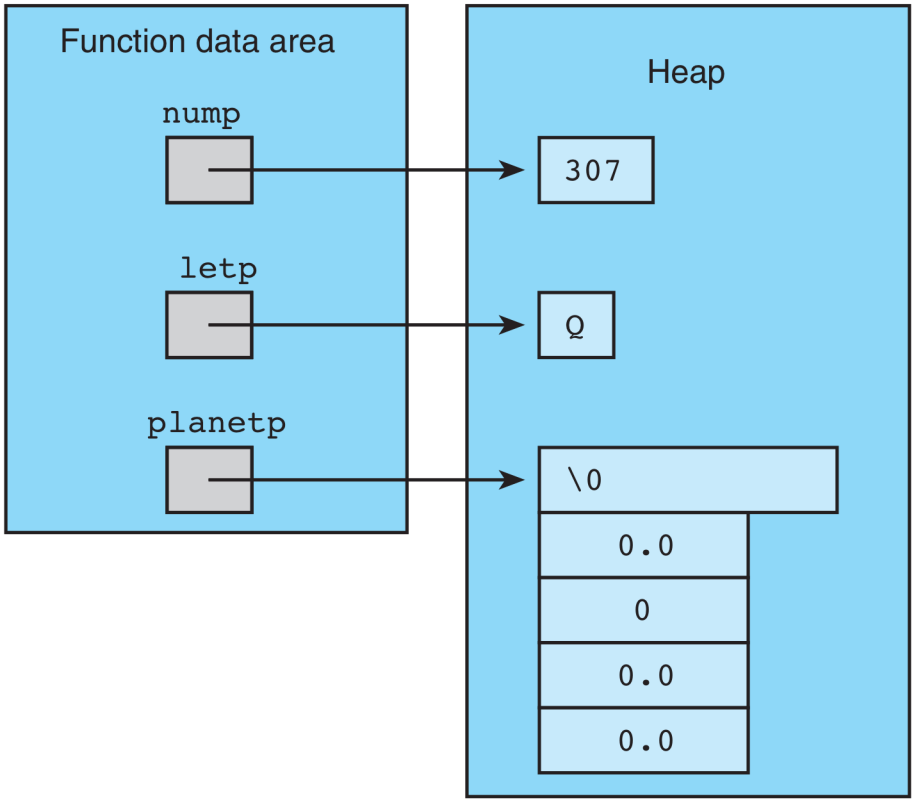
Memory Allocation (2)

```
nump = (int *)malloc(sizeof (int));
letp = (char *)malloc(sizeof (char));
planetp = (planet_t *)malloc(sizeof (planet_t));
```



Memory Allocation (3)

```
planet_t blank_planet = {"", 0, 0, 0, 0};
*nump = 307;
*letp = 'Q';
*planetp = blank_planet;
```



Dynamic Memory Allocation with calloc()

```
1.  #include <stdlib.h>  /* gives access to calloc */
2.  int scan_planet(planet_t *plnp);
3.
4.  int
5.  main(void)
6.  {
7.        char     *string1;
8.        int      *array_of_nums;
9.        planet_t *array_of_planets;
10.       int       str_siz, num_nums, num_planets, i;
11.       printf("Enter string length and string> ");
12.       scanf("%d", &str_siz);
13.       string1 = (char *)calloc(str_siz, sizeof (char));
14.       scanf("%s", string1);
15.
16.       printf("\nHow many numbers?> ");
17.       scanf("%d", &num_nums);
```

Dynamic Memory Allocation with calloc()

```
18.         array_of_nums = (int *)calloc(num_nums, sizeof (int));
19.         array_of_nums[0] = 5;
20.         for (i = 1; i < num_nums; ++i)
21.               array_of_nums[i] = array_of_nums[i - 1] * i;
22.
23.         printf("\nEnter number of planets and planet data> ");
24.         scanf("%d", &num_planets);
25.         array_of_planets = (planet_t *)calloc(num_planets,
26.                                               sizeof (planet_t));
27.         for (i = 0; i < num_planets; ++i)
28.               scan_planet(&array_of_planets[i]);
29.         . . .
30. }

Enter string length and string> 9 enormous

How many numbers?> 4

Enter number of planets and planet data> 2
Earth    12713.5 1  1.0 24.0
Jupiter 142800.0 4 11.9  9.925
```

Memory Functions

Function prototype	Function description
`void *memcpy(void *s1, const void *s2, size_t n);`	*Copies n bytes* from the object pointed to by `s2` into the object pointed to by `s1`. A pointer to the resulting object is returned.
`void *memmove(void *s1, const void *s2, size_t n);`	*Copies n bytes* from the object pointed to by `s2` into the object pointed to by `s1`. The copy is performed as if the bytes were first copied from the object pointed to by `s2` into a *temporary array* and then from the temporary array into the object pointed to by `s1`. A pointer to the resulting object is returned.
`int memcmp(const void *s1, const void *s2, size_t n);`	*Compares the first n bytes* of the objects pointed to by `s1` and `s2`. The function returns `0`, less than `0` or greater than `0` if `s1` is equal to, less than or greater than `s2`.
`void *memchr(const void *s, int c, size_t n);`	*Locates the first occurrence* of `c` (converted to `unsigned char`) in the first `n` bytes of the object pointed to by `s`. If `c` is found, a pointer to `c` in the object is returned. Otherwise, `NULL` is returned.
`void *memset(void *s, int c, size_t n);`	*Copies c* (converted to `unsigned char`) into the *first n bytes* of the object pointed to by `s`. A pointer to the result is returned.

memcpy()

```
1  // Fig. 8.28: fig08_28.c
2  // Using function memcpy
3  #include <stdio.h>
4  #include <string.h>
5
6  int main(void)
7  {
8     char s1[17]; // create char array s1
9     char s2[] = "Copy this string"; // initialize char array s2
10
11    memcpy(s1, s2, 17);
12    printf("%s\n%s\"%s\"\n",
13       "After s2 is copied into s1 with memcpy,",
14       "s1 contains ", s1);
15 }
```

```
After s2 is copied into s1 with memcpy,
s1 contains "Copy this string"
```

memmove()

```
1  // Fig. 8.29: fig08_29.c
2  // Using function memmove
3  #include <stdio.h>
4  #include <string.h>
5
6  int main(void)
7  {
8     char x[] = "Home Sweet Home"; // initialize char array x
9
10    printf("%s%s\n", "The string in array x before memmove is: ", x);
11    printf("%s%s\n", "The string in array x after memmove is: ",
12       (char *) memmove(x, &x[5], 10));
13 }
```

```
The string in array x before memmove is: Home Sweet Home
The string in array x after memmove is: Sweet Home Home
```

memcmp()

```
1  // Fig. 8.30: fig08_30.c
2  // Using function memcmp
3  #include <stdio.h>
4  #include <string.h>
5
6  int main(void)
7  {
8     char s1[] = "ABCDEFG"; // initialize char array s1
9     char s2[] = "ABCDXYZ"; // initialize char array s2
10
11    printf("%s%s\n%s%s\n\n%s%2d\n%s%2d\n%s%2d\n",
12       "s1 = ", s1, "s2 = ", s2,
13       "memcmp(s1, s2, 4) = ", memcmp(s1, s2, 4),
14       "memcmp(s1, s2, 7) = ", memcmp(s1, s2, 7),
15       "memcmp(s2, s1, 7) = ", memcmp(s2, s1, 7));
16 }
```

```
s1 = ABCDEFG
s2 = ABCDXYZ

memcmp(s1, s2, 4) =  0
memcmp(s1, s2, 7) = -1
memcmp(s2, s1, 7) =  1
```

memchr()

```
1  // Fig. 8.31: fig08_31.c
2  // Using function memchr
3  #include <stdio.h>
4  #include <string.h>
5
6  int main(void)
7  {
8     const char *s = "This is a string"; // initialize char pointer
9
10    printf("%s\'%c\'%s\"%s\"\n",
11       "The remainder of s after character ", 'r',
12       " is found is ", (char *) memchr(s, 'r', 16));
13 }
```

```
The remainder of s after character 'r' is found is "ring"
```

memset()

```
1  // Fig. 8.32: fig08_32.c
2  // Using function memset
3  #include <stdio.h>
4  #include <string.h>
5
6  int main(void)
7  {
8     char string1[15] = "BBBBBBBBBBBBBB"; // initialize string1
9
10    printf("string1 = %s\n", string1);
11    printf("string1 after memset = %s\n",
12       (char *) memset(string1, 'b', 7));
13 }
```

```
string1 = BBBBBBBBBBBBBB
string1 after memset = bbbbbbbBBBBBBB
```