CHARUSAT
CHAROTAR UNIVERSITY OF SCIENCE AND TECHNOLOGY

CE143: COMPUTER CONCEPTS & PROGRAMMING

July – November 2019

Chapter – 13

Dynamic Memory Allocation

DEPSTAR
Devang Patel Institute of Advance Technology and Research

Objectives

- To describe dynamic memory allocation
- To differentiate between malloc and calloc
- To explain how allocated space is altered or released

Introduction

In this chapter, we will discuss

- Memory allocation process
- Allocating a block of memory: MALLOC
- Allocating multiple blocks of memory: CALLOC
- Releasing the used space: FREE
- Altering the size of a block: REALLOC

Need for Dynamic Memory Allocation

- Data can be dynamic in nature i.e. number of data items keep changing during execution of a program.
- Example: Consider a program for processing the list of customers of corporation.
  - the list grows when names are added and shrinks when names are deleted.
- Dynamic data structures provide flexibility in adding, deleting or rearranging data items at runtime.
- It allows us to allocate additional memory space or to release unwanted space at runtime.

Memory Allocation Process

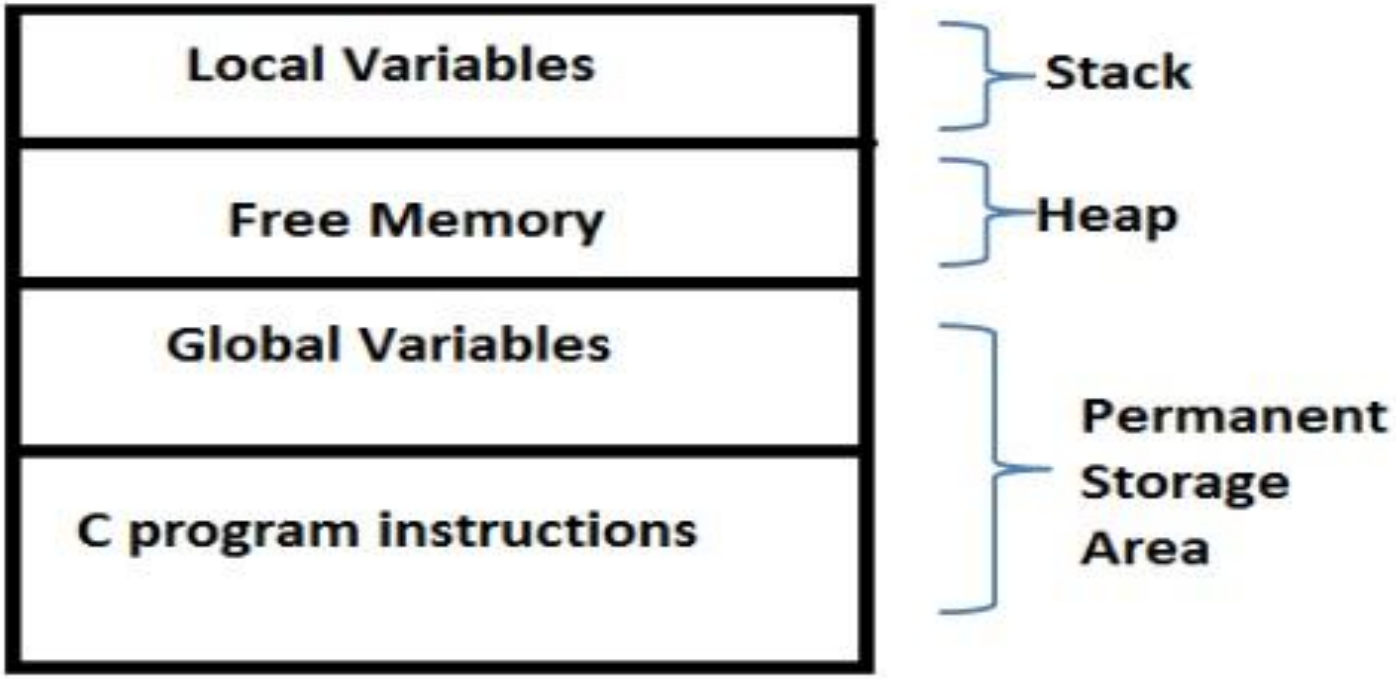


Fig: Storage of a C program

Memory Allocation Process

- **Permanent Storage Area:** The program instructions and global and static variables are stored in this region.

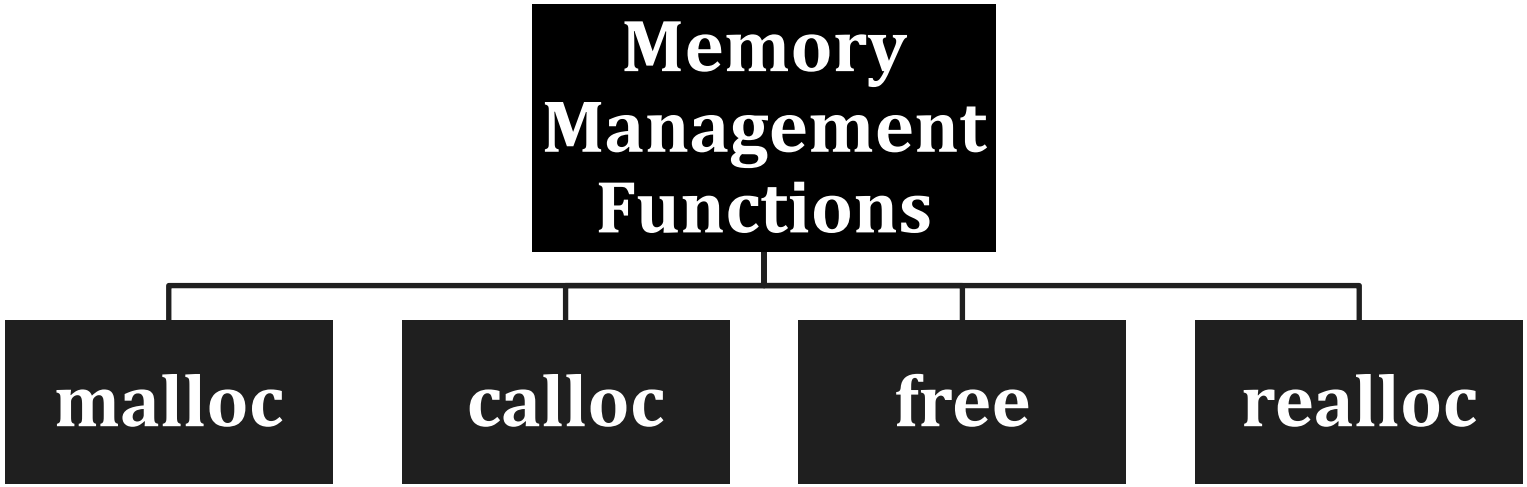
- **Stack:** local variables are stored.

- **Heap:** the memory allocation s[ace located between these two regions is available for dynamic allocation during execution of the program. This free memory region is called heap.

Note: Size of heap keeps changing during the program execution.

Dynamic Memory Allocation

- The process of allocating memory at runtime is known as dynamic memory allocation.



Allocating a Block of Memory: MALLOC

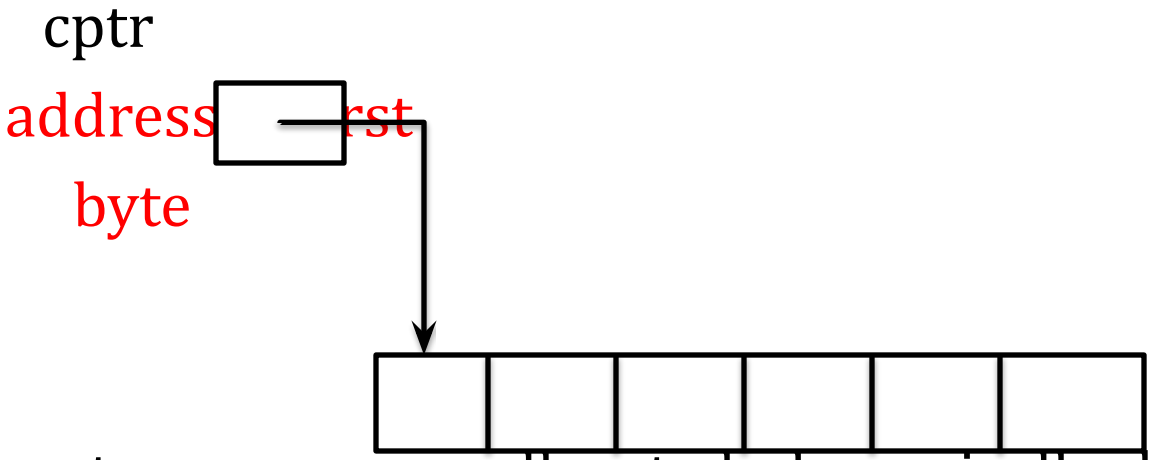
- It reserves a block of memory of specified size and returns a pointer of type void.

  ptr = (cast-type*) malloc(byte-size);

- **ptr** pointer of type cast-type
- **malloc** returns a pointer to an area of memory with size *byte-size*
- Example: x=(int *) malloc (100 *sizeof(int))
- A memory space equivalent to “100 times the size of an int” bytes is reserved and the address of the first byte of the memory allocated is assigned to the pointer x of type int.

Allocating a Block of Memory: MALLOC

- cptr = (char*) malloc(10) , this allocates 10 byte of space for the pointer cptr of type char.



- The storage space allocated dynamically has no name, therefore its contents can be accessed only through a pointer.

Allocating a Block of Memory: MALLOC

- We may also use malloc to allocate space for complex data types such as structures.
- **Example:**

  st_var = (struct store*) malloc(sizeof(structure))
- st_var pointer of type struct store
- malloc allocates a block of contiguous bytes.
- The allocation can fail if the space in the heap is not sufficient.
- If it fails it returns NULL.

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

void main()
{
   int *p, *table;
   int size;
   printf("\nWhat is the size of table?");
   scanf("%d",&size);
   printf("\n");

   if((table = (int*)malloc(size *sizeof(int))) == 0)
     {
        printf("No space available \n");
        exit(1);
     }
   printf("\n Address of the first byte is  %u \n", table);
   /* Reading table values*/
   printf("\nInput table values\n");

   for (p=table; p<table + size; p++)
       scanf("%d",p);
   /* Printing table values <span id="IL_AD3" class="IL_AD">in reverse</span> order*/
   for (p = table + size -1; p >= table; p --)
     printf("%d is stored at address %u \n",*p,p);

getch();
}
```

Output:

```
What is the size of the table? 5
Address of the first byte is 2262
Input table values
11 12 13 14 15
15 is stored at address 2270
14 is stored at address 2268
13 is stored at address 2266
12 is stored at address 2264
11 is stored at address 2262
```

Allocating Multiple Blocks of Memory: CALLOC

- It is used for requesting memory space at runtime for storing derived data types such as arrays and structures.
- Allocates multiple blocks of storage, each of same size and sets all bytes to zero.
- **General form:**

  ptr = (cast-type*) calloc(n,elem-size);
- This allocates contiguous space for n blocks, each of size elem-size bytes.
- All bytes are initialized to *zero.*
- A pointer to the first byte of the allocated region is returned.

Example

```
struct student
{
      char name[25];
      float age;
      long int id_name;
}
typedef struct student record;
record *st_ptr;
int class_size=30;
st_ptr = (record*) calloc(class_size, sizeof(record));
---------
---------
```

record is of type struct student having 3 members

calloc allocates memory to hold 30 such records

Example

- To check that requested memory has been allocated successfully or not before using the st_ptr, following can be done:

```
if(st_ptr == NULL)
{
    printf("Memory not sufficient");
    exit(1);
}
```

Releasing the used space: FREE

- It is required to release the space when it is not required. This can be done using free function:
- General Form:

```
free(ptr);
```

- Ptr pointer to a memory block, which has already been created by malloc or calloc.

Note: It is not the pointer that is being released but rather what it points to.

Altering the size of a Block: REALLOC

- If previously allocated memory is not sufficient and we need additional space for elements.
- If memory allocated is much larger than required so we want to reduce it.
- In both cases, we can change the memory size with the help of realloc, which is called *reallocation of memory.*
- Example:

```
ptr =malloc(size);
ptr = realloc(ptr, newsize);
```

- Allocates a new memory space of size newsize to the pointer variable ptr and returns a pointer to the first byte of the new memory block.

Altering the size of a Block: REALLOC

- The new memory block may or may not begin at the same place as the old one.
- If it fails in locating additional space, it returns NULL pointer and the original block is lost.

```c
#include <stdio.h>
#include<stdlib.h>
#define NULL 0
main()
{
     char *buffer;
     /* Allocating memory */
if((buffer = (char *)malloc(10)) == NULL)
     {
          printf("malloc failed.\n");
          exit(1);
     }
     printf("Buffer of size %d created \n",_msize(buffer));
     strcpy(buffer, "HYDERABAD");
     printf("\nBuffer contains: %s \n ", buffer);
     /* Realloction */
if((buffer = (char *)realloc(buffer, 15)) == NULL)
     {
          printf("Reallocation failed. \n");
          exit(1);
     }
     printf("\nBuffer size modified. \n");
     printf("\nBuffer still contains: %s \n",buffer);
     strcpy(buffer, "SECUNDERBAD");
     printf("\nBuffer now contains: %s \n",buffer);
/* Freeing memory */

free(buffer);
 }
```

```
Output
Buffer of size 10 created
Buffer contains: HYDERABAD
Buffer size modified
Buffer still contains: HYDERABAD
Buffer now contains: SECUNDERABAD
```

Previous Year Questions

- Explain malloc() function
- Which of the following is the correct syntax to allocate memory at runtime for 5 integer using malloc ( ) function, if ptr is integer pointer?

  a) ptr = (void *)malloc(5, sizeof(int));

  b) ptr = (int *) malloc(5*size(int));

  c) ptr = (int *)malloc(5*sizeof(int));

  d) ptr = malloc(5*sizeof(int));
- Differentiate dynamic memory allocation and static memory allocation of variable.
- Explain realloc() with syntax and example.
- The return type of malloc() function is_____.
- Which function is used to de-allocate the memory occupied at run-time?

  (A) malloc (B) realloc (C) free (D) realloc
- An array created using malloc() at run time is referred to as_____.