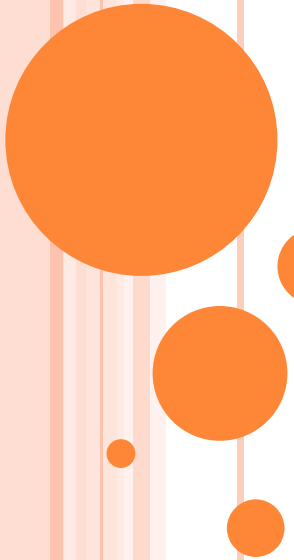


Dynamic Memory Allocation in C

Spurthi N Anjan

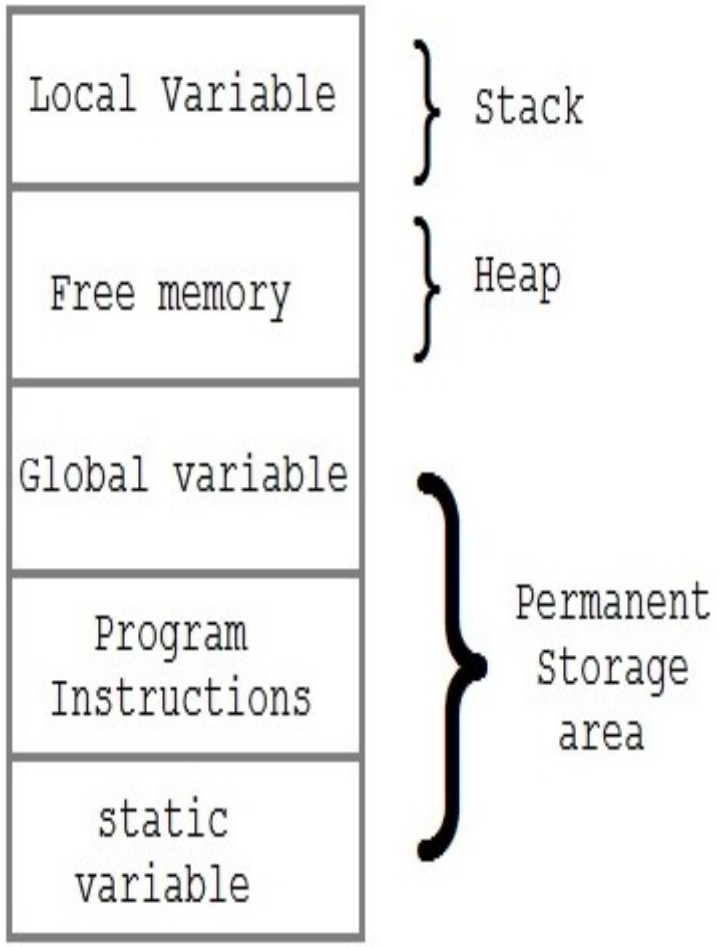
Dept of CSE, PESU

MEMORY ALLOCATION

- The blocks of information in a memory system is called memory allocation.
- To allocate memory it is necessary to keep in information of available memory in the system.
- If memory management system finds sufficient free memory, it allocates only as much memory as needed, keeping the rest available to satisfy future request.
- In memory allocation has two types. They are
  - Static memory allocation.
  - Dynamic memory allocation.

Memory Allocation Process

- Global variables, static variables and program instructions get their memory in permanent storage area whereas local variables are stored in a memory area called Stack.
- The memory space between these two region is known as Heap area.
- This region is used for dynamic memory allocation during execution of the program. The size of heap keep changing.



Memory Allocation Process

- All the program instructions, global and static variables are stored in the memory region called the permanent storage area.
- Local variables are stored in the memory region called the stack.
- The free memory area between the permanent storage area and stack is called the heap. This is the area available for the dynamic allocation during the execution of the program.
- The size of the heap keeps changing during the program execution due to the creation and removable of variables that are local to functions and blocks.
- So, a common problem we can encounter is overflow of memory during dynamic memory allocation.
- The above mentioned memory allocation functions returns NULL when there is no sufficient memory space for allocation.

Memory Allocation Process

- Memory can be allocated for variables using different techniques

1. static allocation
   - decided by the compiler
   - allocation at compile time
2. automatic allocation
   - decided by the compiler
   - allocation at run time
   - allocation on entry to the block and deallocation on exit
3. dynamic allocation
   - code generated by the compiler
   - allocation and deallocation on call to memory allocation functions: malloc, calloc, realloc.

MEMORY ALLOCATION

- STATIC MEMORY ALLOCATION
  - In static memory allocation, size of the memory may be required for the that must be define before loading and executing the program.
- DYNAMIC MEMORY ALLOCATION
  - In the dynamic memory allocation, the memory is allocated to a variable or program at the run time.
  - The only way to access this dynamically allocated memory is through pointer.

WHY DYNAMIC MEMORY ALLOCATION?

- Usually, so far, the arrays and strings we're using have fixed length (i.e., length is known at compile time)
- Example:

char myStr[ ] 11 ; // allocates memory for 10 chars

printf("Enter a string:\n");

scanf("%s", myStr);

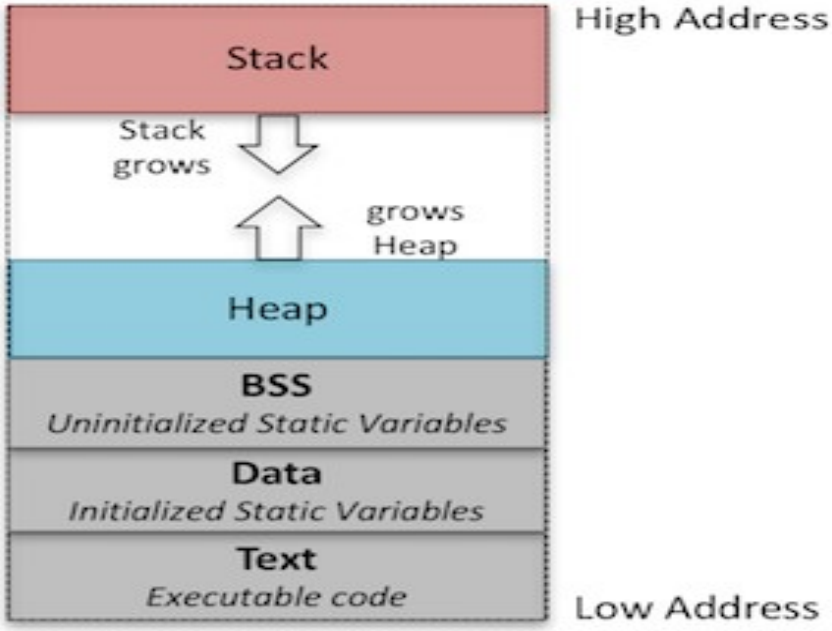
- What if the user wants to enter a string more than 10 chars long or if the length is known only at run time?

What Is Dynamic Memory?

- Memory allocation is a very important part of software development.
- When the program is loaded into the system memory, the memory region allocated to the program is divided into three broad regions: stack, heap, and code.
- Stack region is used for storing program's local variables when they're declared.
- Also, variables and arrays declared at the start of a function, including main, are allocated stack space. Stacks grow from high address to low address.
- Heap region is exclusively for dynamic memory allocation. Unlike stacks, heaps grow from low address to high address.

What Is Dynamic Memory?

- Code region can be further divided as follows:
  - BSS segment: stores uninitialized static variables
  - Data segment: stores static variables that are initialized
  - Text segment: stores the program's executable instructions



Program Memory Layout

C Dynamic Memory Allocation

- An array is a collection of a fixed number of values. Once the size of an array is declared, you cannot change it.
- Sometimes the size of the array you declared may be insufficient. To solve this issue, you can allocate memory manually during run-time. This is known as dynamic memory allocation in C programming.
- To allocate memory dynamically, library functions are **malloc(), calloc(), realloc() and free()** are used.
- These functions are defined in the <stdlib.h> header file.

DIFFERENCE BETWEEN STATIC MEMORY ALLOCATION AND DYNAMIC MEMORY ALLOCATION.

static memory allocation	**dynamic memory allocation**
memory is allocated at compile time.	memory is allocated at run time.
memory can't be increased while executing program.	memory can be increased while executing program.
used in array.	used in linked list.

Static vs Dynamic Memory Allocation

Static memory allocation is a method of allocating memory, and once the memory is allocated, it is fixed.	Dynamic memory allocation is a method of allocating memory, and once the memory is allocated, it can be changed.
Modification	
In static memory allocation, it is not possible to resize after initial allocation.	In dynamic memory allocation, the memory can be minimized or maximize accordingly.
Implementation	
Static memory allocation is easy to implement.	Dynamic memory allocation is complex to implement.
Speed	
In static memory, allocation execution is faster than dynamic memory allocation.	In dynamic memory, allocation execution is slower than static memory allocation.
Memory Utilization	
In static memory allocation, cannot reuse the unused memory.	Dynamic memory allocation allows reusing the memory. The programmer can allocate more memory when required .He can release the memory when necessary.

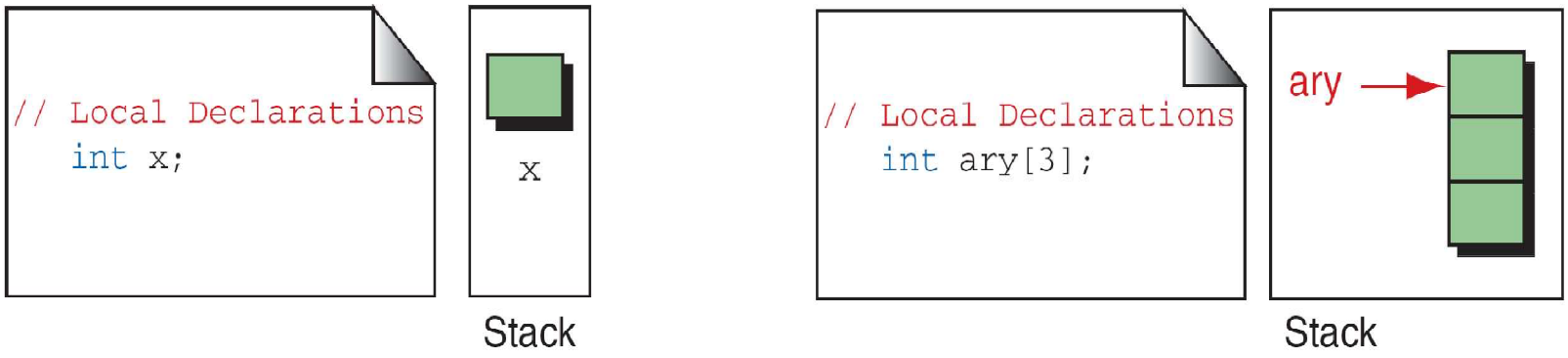
Spurthi N Anjan
Dept of CSE, PESU

METHODS USED FOR DYNAMIC MEMORY ALLOCATION

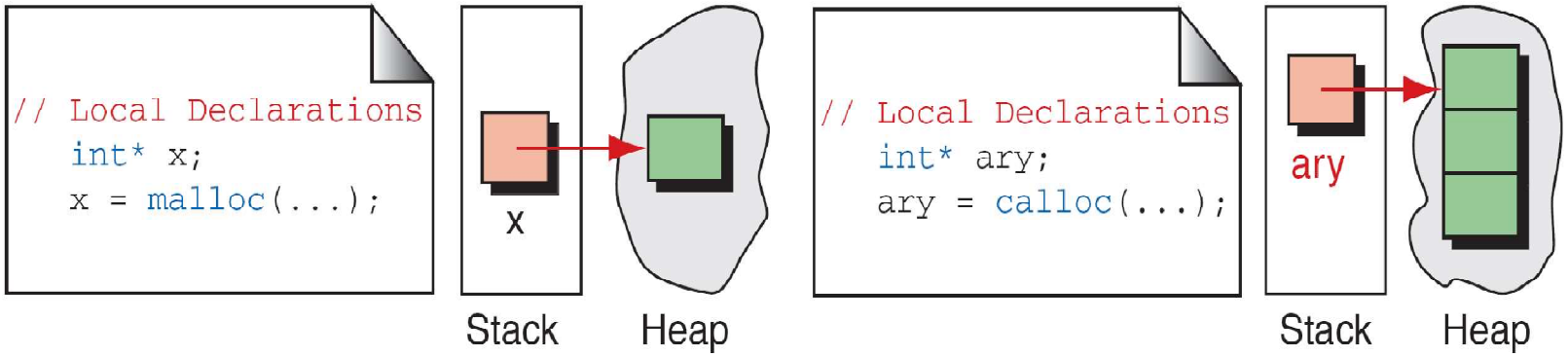
malloc()	Allocates the memory of requested size and returns the pointer to the first byte of allocated space.
calloc()	Allocates the space for elements of an array. Initializes the elements to zero and returns a pointer to the memory.
realloc()	It is used to modify the size of previously allocated memory space.It reallocates the memory occupied by malloc() or calloc() functions.
free()	frees the dynamically allocated memory.

Spurthi N Anjan
Dept of CSE, PESU

DYNAMIC MEMORY ALLOCATION



(a) Static Memory Allocation



(b) Dynamic Memory Allocation

C MALLOC()

- The name "malloc" stands for memory allocation.
- The malloc() function reserves a block of memory of the specified number of bytes.
- The malloc() function allocates single block of requested memory.
- It doesn't initialize memory at execution time, so it has garbage value initially.
- It returns a pointer of void which can be casted into pointers of any form.

Syntax of malloc()

ptr = (castType*) malloc(size);

Example : ptr = (float*) malloc(100 * sizeof(float));

C MALLOC()

Syntax of malloc()

ptr = (castType*) malloc(size);

Example : ptr = (float*) malloc(100 * sizeof(float));

- The above statement allocates 400 bytes of memory. It's because the size of float is 4 bytes.
- Input is the number of consecutive bytes to be allocated.
- And, the pointer ptr holds the address of the first byte in the allocated memory.
- The expression results in a NULL pointer if the memory cannot be allocated.
- To use malloc(), you must #include<stdlib.h>.

C MALLOC()

malloc()

```
char *charP;        /* declare a pointer to char */
```


```
charP = malloc(10);
```

charP contains the address of the beginning of that block.

MALLOC ALLOCATES BYTES

- The malloc function reserves a block of memory of specified size and returns a pointer of type void. This means that we can assign it to any type of pointer.
- If you want a character array that stores 10 characters (including '\0'):

char *p = malloc(10);

- If you want to allocate storage for 10 ints (or doubles or floats), you can't do this:

int *p = malloc(10); /* WRONG! Why? */

ALLOCATE INT AND DOUBLE ARRAY

```
int *intP;

double *doubleP;
```

- // Allocate space for 10 integers

```
intP = malloc(10 * sizeof(int));

// Allocates 40 bytes, sizeof(int) = 4
```

- // Allocate space for 10

```
doubles doubleP = malloc(10 * sizeof(double));

// Allocates 80 bytes, sizeof(double) = 8
```

C CALLOC()

- The name "calloc" stands for contiguous allocation.
- The malloc() function allocates memory and leaves the memory uninitialized.

Whereas, the calloc() function allocates memory and initializes all bits to zero.

- It returns NULL if memory is not sufficient.

Syntax of calloc()

ptr = (castType*)calloc(n, size);

Example: ptr = (float*) calloc(25, sizeof(float));

- The above statement allocates contiguous space in memory for 25 elements of type float.

C CALLOC()

- We use the calloc function to allocate memory at run time for derived data types like arrays and structures.
- Using calloc function we can allocate multiple blocks of memory each of the same size and all the bytes will be set to 0.
- This is different from the malloc which is used to allocate single block of memory space.

C FREE()

- Dynamically allocated memory created with either calloc() or malloc() doesn't get freed on their own.
- You must explicitly use free() to release the space.

Syntax of free()

free(ptr);

- where ptr “points to” memory previously previously allocated allocated by malloc() function .
- This statement frees the space allocated in the memory pointed by ptr.

C REALLOC()

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

Syntax of realloc()

ptr = realloc(ptr, x);

- Here, ptr is reallocated with a new size x.
- If memory is not sufficient for malloc() or calloc(), you can reallocate the memory by realloc() function. In short, it changes the memory size.

C REALLOC()

- realloc takes a pointer to allocated memory and reallocates the memory to a larger size.
- If it can make the old block bigger, great.
- If not, it will get another, larger block, copy the old contents contents to the new contents contents, free the old contents contents and return a pointer to the new.

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

intP = realloc(intP, 2*sizeof(intP));
```

- intP may be different after a realloc!

C REALLOC()

- Following are the points to note when using realloc function.
- The functions allocates a new memory space of size new_size and returns a pointer that points at the first address of the newly reallocated memory space.
- The new_size may be larger than or less than the previously allocated memory space.
- The newly reallocated memory block may not start from the same location as the old allocated space.
- If the relloacted memory size is bigger than the original size and if there is no space to accommodate the new size in the same region then, this function will allocate the new memory block elsewhere, and will move the content of the old block to the new location.
- If this function fails to get the new size memory location then it will return NULL.

CALLOC VS. MALLOC: DIFFERENCES

- The calloc function is generally more suitable and efficient than that of the malloc function.
- While both the functions are used to allocate memory space, calloc can allocate multiple blocks at a single time. You don't have to request for a memory block every time.
- The calloc function is used in complex data structures which require larger memory space.
- The memory block allocated by a calloc function is always initialized to zero while in malloc it always contains a garbage value.

CALLOC VS. MALLOC: DIFFERENCES

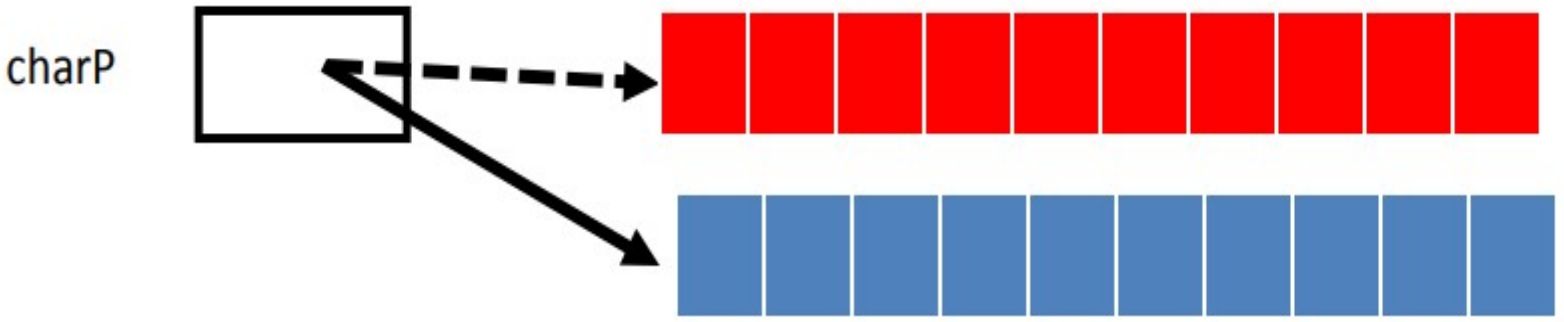
malloc()	calloc()
int *ptr; ptr = malloc(20 * sizeof(int)); For the above, 20*4 bytes of memory only allocated in one block. Total = 80 bytes	int *ptr; Ptr = calloc(20, 20 * sizeof(int)); For the above, 20 blocks of memory will be created and each contains 20*4 bytes of memory. Total = 1600 bytes
malloc () doesn't initializes the allocated memory. It contains garbage values	calloc () initializes the allocated memory to zero
type cast must be done since this function returns void pointer int *ptr; ptr = (int*)malloc(sizeof(int)*20);	Same as malloc () function int *ptr;ptr = (int*)calloc(20, 20 * sizeof(int));

MEMORY LEAK

- Memory leaks refer to memory that has been allocated by an application, but not properly released back once that memory is no longer needed.
- Many systems have multiple applications running on their systems and programming errors usually result in “memory leaks”.
- Memory leaks may not affect the functionality of the application, but left unattended in a system, memory leaks can cause the machine to crash.
- This is why, servers restart often to avoid them to from going down. Programmers typically allocate memory and then somehow may lose the reference to that memory block.

MEMORY LEAK

- If malloc'ed memory is not free'ed, then the OS will "leak memory"
- This means that memory is allocated to the program but not returned to the OS when it is finished using it
- The program program therefore therefore grows larger over time and eventually runs out of memory!



If you don't free the allocated memory, previous block is still "ours" according to the OS, but we can no longer find it (no pointer to it). That block is an orphan!

- **Advantages of Dynamic memory allocation**

- Data structures can grow and shrink according to the requirement.

- We can allocate (create) additional storage whenever we need them.

- We can de-allocate (free/delete) dynamic space whenever we are done with them.

- Dynamic Allocation is done at run time.

- **Disadvantages of Dynamic memory allocation**

- As the memory is allocated during runtime, it requires more time.

- Memory needs to be freed by the user when done. This is important as it is more likely to turn into bugs that are difficult to find.

NOTE:

- We can refer to memory allocated in the heap only through a pointer.
- Using a pointer after its memory has been released is a common programming error. Guard against it by clearing the pointer.
- The pointer used to free memory must be of the same type as the pointer used to allocate the memory.
- If the space in memory allocated by malloc is insufficient, then the allocation fails and malloc returns NULL pointer.
- calloc initializes the allocated memory to zero value whereas malloc doesn't.
- calloc is used to allocate memory to mostly arrays and structures

SUMMARY

- We can dynamically manage memory by creating memory blocks as needed in the heap
- In dynamic memory allocation, memory is allocated at a run time.
- Dynamic memory allocation permits to manipulate strings and arrays whose size is flexible and can be changed anytime in your program.
- It is required when you have no idea how much memory a particular structure is going to occupy.

SUMMARY

- dynamic memory allocation function which stands for memory allocation that blocks of memory with the specific size initialized to a garbage value
- Calloc is a contiguous memory allocation function that allocates multiple memory blocks at a time initialized to 0
- Realloc is used to reallocate memory according to the specified size.
- Free function is used to clear the dynamically allocated memory.

EXERCISES

- C program to create memory for **int, char and float variable at run time**.
- C program to **input and print text using Dynamic Memory Allocation**.
- Write a program to find the largest number in an array .
- Write a program to sort the strings in alphabetical order.
- Write a 'C' program to accept 'n' numbers from user and sort them in ascending order using Dynamic Memory Allocation.
- WAP to make a copy of string into a dynamically allocated character array of the appropriate size
- Write a 'C' program to reverse an array's elements using dynamic memory allocation
- C program to find sum of array elements using Dynamic Memory Allocation.

EXERCISES

- Write a program in C to dynamically allocate memory using malloc function to store N integer numbers entered by the user and then print the sum.
- Write a program in C to dynamically allocate memory using calloc function for a structure.
- Write a program in C to reallocate previously allocated memory space.
- C program to read and print the student details using structure and Dynamic Memory Allocation.
- C program to read and print the N student details using structure and Dynamic Memory Allocation.