

Lecture 4

Dynamic Memory Allocation

Dynamic Memory Allocation

The process of allocating memory at run time is known as dynamic memory allocation.

C does not inherently have this facility, there are four library routines known as “memory management functions” that can be used for allocating and freeing memory during program execution.

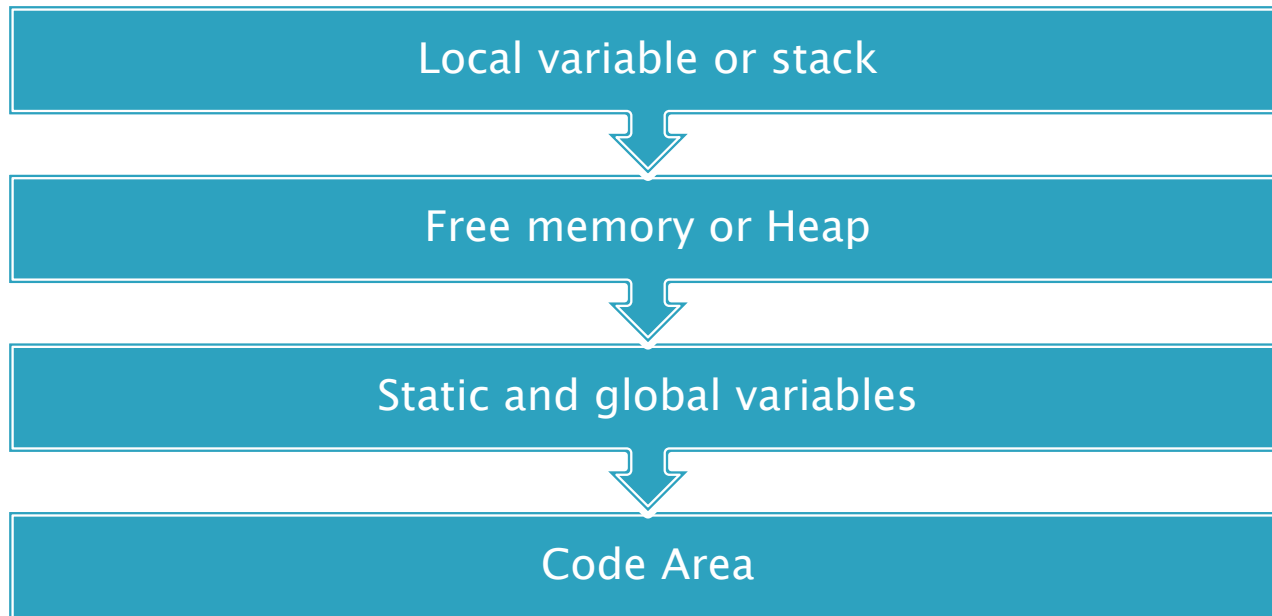
These functions are:

1. `malloc()`
2. `Calloc()`
3. `Realloc()`
4. `Free()`

These functions are defined either in the header file “alloc.h” or in `stdlib.h` or in both

Memory Allocation Process

Memory allocation process associated with a C program. The RAM can be divided into 4 areas.



Memory Allocation Process

Memory allocation process associated with a C program. The RAM can be divided into 4 areas.

1. *Local variables are stored in area called stack.*
1. *The code area is the region in the RAM where your C program instructions, after they are translated into machine language are stored.*
1. *There is a separate area for storing the global and static variables.*
1. *The free memory area is called the heap. The size of heap keeps changing when program is executed due to creation and **deletion** of variables.*

Memory Allocation Process

1) malloc in C :

The function most commonly used for dynamic memory allocation is malloc(). The syntax of the function is

malloc(x);

Where x is an unsigned integer which stands for the number of bytes you want to draw from the heap. The function returns a pointer, if your request is successful. Otherwise malloc return a void pointer.

If you want it to point to any datatype of data you should write

(data_tpye*)malloc(x);

Memory Allocation Process

- 1) **malloc in C :**
consider the following statement

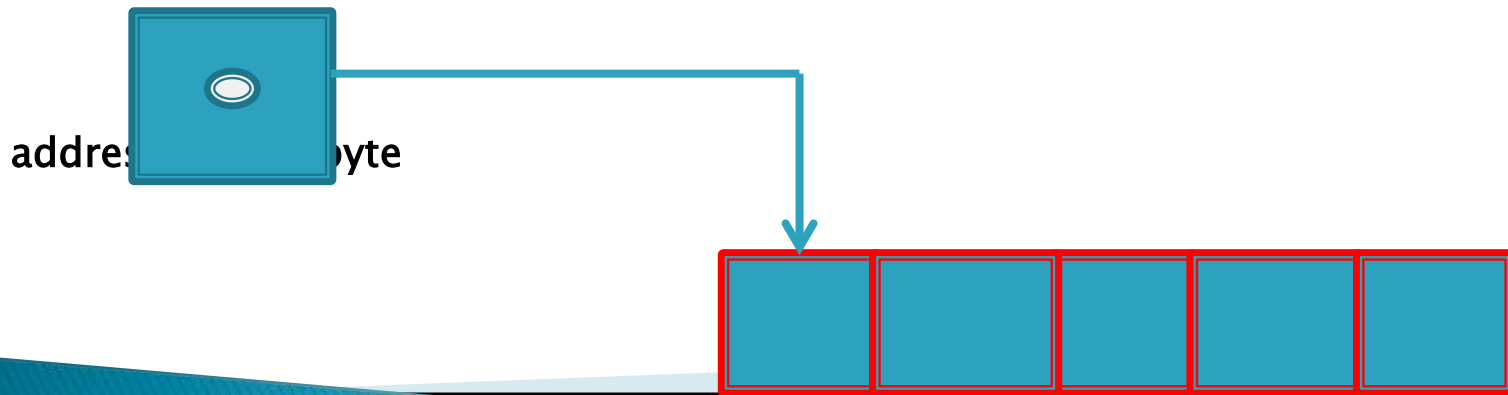
```
ptr=(int*)malloc(sizeof(int));
```

In this example ,a memory space equivalent to size of an `int` byte is reserved and the address of first byte of memory allocated is assigned to the pointer `ptr` of type of `int`.

e.g.

```
ptr= (char*) malloc(5);
```

Allocate 5 byte of space for the pointer `ptr` of type `char` as shown
`ptr`



We can also use malloc to allocate memory for complex data type such as structures

Sample Program : malloc.c

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
void main()
{
int a,*ptr;
a=10;
ptr=(int*)malloc(a*sizeof(int));
ptr=a;
printf("%d",ptr);
free(ptr);
getch();
}
```

Program Explanation :

1.**#include<stdio.h>** header file is included because, the C in-built statement **printf** we used in this program comes under **stdio.h** header files.

2.**#include<conio.h>** is used because the C in-built function **getch()** comes under **conio.h** header files.

3.**stdlib.h** is used because **malloc()** comes under it.

4.**int** type variable **a** and pointer ***ptr** are declared.

5.Variable **a** is assigned a value of **10**.

6.**malloc()** is used to allocate memory to pointer **ptr**.

7.The size given through **malloc()** to **ptr** is **sizeof(int)**.

8.Now **ptr** is assigned the value of **a**. **ptr=a**; so the value **10** is assigned to **ptr**, for which, we dynamically allocated memory space using **malloc**.

9.The value in **ptr** is displayed using **printf**

10.Then allocated memory is **freed** using the C in-built function **free()**.

Memory Allocation Process

1) Calloc in C :

Function `calloc()`, like `malloc()` allocates memory in a dynamic manner. It takes, as arguments, two values

`p=(t*)calloc(n,b)`

the above statement allocates `n` blocks of memory, each block with `b` bytes. If there is not enough space, a NULL pointer is returned.

this function is useful for storing arrays. If each element of an array requires `b` bytes and the array is required to store `K` elements we can allocate memory by the following statement

`p=(t*)calloc(k,b)`

Syntax :

```
pointer=(data_type*)calloc(no of memory blocks, size of each block in bytes);
```

Sample Program : calloc.c

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
void main()
{
int *ptr,a[6]={ 1,2,3,4,5,6};
int i;
ptr=(int*)calloc(a[6]*sizeof(int),2);
for(i=0;i<7;i++)
{
printf("\n %d",*ptr+a[i]);
}

free(ptr);
getch();
}
```

Explanation :

1. **#include<stdio.h>** header file is included because, the C in-built statement **printf** we used in this program comes under **stdio.h** header files.

2. **#include<conio.h>** is used because the C in-built function **getch()** comes under **conio.h** header files.

3. **stdlib.h** is used because **malloc()** comes under it.

4. A Pointer ***ptr** of type **int** and array **a[6]** is declared.

5. Array **a[6]** is assigned with **6** values.

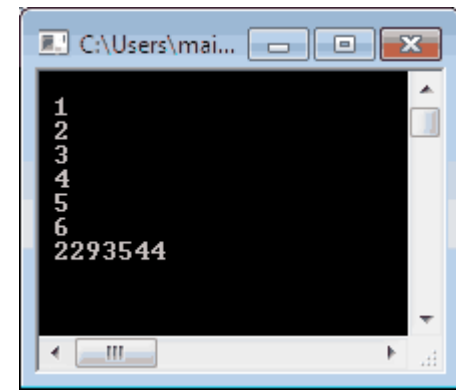
6. Another variable **i** of type **int** is declared.

7. **Calloc()** is used to allocate memory for **ptr**. **6** blocks of memory is allocated with each block having **2** bytes of space.

8. Now variable **i** is used in **for** to cycle the loop **6 times** on **incremental** mode.

9. On each cycle the data in allocated memory in **ptr** is printed using ***ptr+a[i]**.

10. Then the memory space is **freed** using **free(ptr)**.



```
C:\Users\mai...  
1  
2  
3  
4  
5  
6  
 2293544
```

Memory Allocation Process

1) **realloc in C :**

we can change the memory size already allocated with the help of the function `realloc`.

This process is called the reallocation of the memory. For example, if the original allocation is done on the statement

`ptr = malloc(size)`

Then the reallocation of the space may be done by the statement

`ptr = realloc(ptr, newsize)`

Memory Allocation Process

1) **Free()in C :**

the memory must be returned to the heap, when it is no longer required. This is done with free() function . The syntax of the statement is

free (ptr)

Where ptr is the pointer to a block of memory which has been allocated from the heap on request. The function is void in nature and does not return anything.