**Dynamic Memory Allocation in C**

The process of allocating memory at runtime is known as **dynamic memory allocation**. Library routines known as **memory management functions** are used for allocating and freeing memory during execution of a program. These functions are defined in **stdlib.h** header file.

Note :

pointers are used for dynamic(happens at runtime) memory location.

While allocating memory we have to mention what type(datatype) of element.

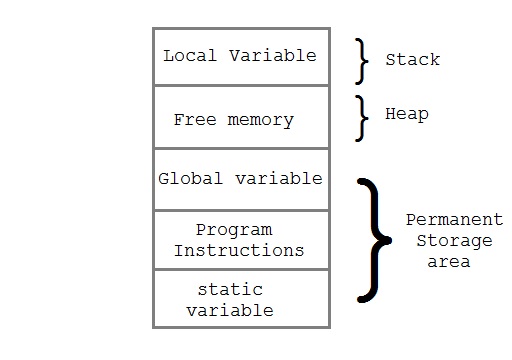
**List of functions found in** **stdlib.h**

|  |  |
| --- | --- |
| **Function** | **Description** |
| malloc() | allocates requested size of bytes and returns a void pointer pointing to the first byte of the allocated space |
| calloc() | allocates space for an array of elements, initialize them to zero and then returns a void pointer to the memory |
| free | releases previously allocated memory |
| realloc | modify the size of previously allocated space |

**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.



**Allocating block of Memory**

**Malloc**

malloc() function is used for allocating block of memory at runtime. This function reserves a block(unlike array elements no boundary for individual elements) of memory of the given size and returns a **pointer** of type void.

This means that we can assign it to any type of pointer using typecasting. If it fails to allocate enough space as specified, it returns a NULL pointer.

**Syntax:**

datatype\* malloc(byte-size)

**Example: malloc()**

int \*x;  
x = (int\*) malloc(50 \* sizeof(int)); //memory space allocated to variable x  
free(x); //releases the memory allocated to variable x

**Calloc**

calloc() is another memory allocation function that is used for allocating memory at runtime.

calloc function is normally used for allocating memory to derived data types such as **arrays** and **structures(need contiguous memory locations)**. If it fails to allocate enough space as specified, it returns a NULL pointer.

**Syntax:**

void \*calloc(number of items, element-size)

**Example: calloc()**

int i, n;  
 int \*element;  
 printf("Enter total number of elements: ");  
 scanf("%d", &n);  
 element = (int\*) calloc(n , sizeof(int));

**Realloc**

realloc() changes memory size that is already allocated dynamically to a variable.

**Syntax:**

void\* realloc(pointer, new-size)

Time for an Example: realloc()

int \*x;  
x = (int\*)malloc(50 \* sizeof(int));  
x = (int\*)realloc(x,100); //allocated a new memory to variable x, increased to 100

**Difference between malloc() and calloc()**

|  |  |
| --- | --- |
| **calloc()** | **malloc()** |
| calloc() initializes the allocated memory with 0 value. | malloc() initializes the allocated memory with garbage values. |
| Number of arguments is 2 | Number of argument is 1 |
| **Syntax :**  (cast\_type \*)calloc(blocks , size\_of\_block); | **Syntax :**  (cast\_type \*)malloc(Size\_in\_bytes); |

**Malloc Example**

**sum of list of elements**

#include <stdio.h>

#include <stdlib.h>

int main()

{

int num, i, \*ptr, sum = 0;

printf("Enter number of elements: ");

scanf("%d", &num);

**ptr = (int\*) malloc(num \* sizeof(int));**  //memory allocated using malloc, ptr hold the first address

//check if there is no sufficient memory available

if(ptr == NULL)

{

printf("Error! memory not allocated.");

exit(0);

}

//see the default values here

//for(i=0;i<num;i++){

// printf("%d",\*(ptr+i));

// }

printf("Enter elements of array: ");

for(i = 0; i < num; ++i)

{

scanf("%d", ptr + i);

sum+= \*(ptr + i);

}

printf("Sum = %d", sum);

free(ptr);

/\*

there is no way to check whether memory freed or not, but make sure pointer is assigned null

Freed - available for allocation

\*/

ptr = NULL;

//this gives error

// sum = 0;

// for(i = 0; i < num; ++i){

// scanf("%d", ptr + i);

// sum += \*(ptr + i);

// }

// printf("Sum = %d", sum);

return 0;

}

**Example for Calloc**

**sum of list of elements**

#include <stdio.h>

#include <stdlib.h>

int main()

{

int i, n;

int \*element;

printf("Enter total number of elements: ");

scanf("%d", &n);

/\*

returns a void pointer(which is type-casted to int\*)

pointing to the first block of the allocated space

\*/

element = (int\*) calloc(n,sizeof(int));

/\*

If it fails to allocate enough space as specified,

it returns a NULL pointer.

\*/

if(element == NULL)

{

printf("Error.Not enough space available");

exit(0);

}

int sum=0;

for(i = 0; i < n; i++)

{

scanf("%d", element+i);

sum += \*(element+i);

}

printf("sum of elements is %d", sum);

free(element);

element = NULL;

//error- cant acces after freed memory

// sum=0;

// for(i = 0; i < n; i++)

// {

// scanf("%d", element+i);

// sum += \*(element+i);

// }

return 0;

}

**Malloc and calloc both can be used to allocate memory to user defined datatypes like structures, but it is recommended to use calloc for user defined datatypes.**

#include <stdio.h>

#include <stdlib.h>

struct course

{

int marks;

char subject[30];

};

int main()

{

struct course \*ptr;

int i, noOfRecords;

printf("Enter number of records: ");

scanf("%d", &noOfRecords);

// Allocates the memory for noOfRecords structures with pointer ptr pointing to the base address.

//ptr = (struct course\*) malloc (noOfRecords \* sizeof(struct course));

ptr = (struct course\*) calloc (noOfRecords, sizeof(struct course));

for(i = 0; i < noOfRecords; i++)

{

printf("Enter name of the subject and marks respectively:\n");

scanf("%s %d", &(ptr+i)->subject, &(ptr+i)->marks);

}

printf("Displaying Information:\n");

for(i = 0; i < noOfRecords ; i++)

printf("%s\t%d\n", (ptr+i)->subject, (ptr+i)->marks);

return 0;

}