CHARUSAT
CHAROTAR UNIVERSITY OF SCIENCE AND TECHNOLOGY

Chandubhai S Patel Institute of Technology

Dynamic Memory Management

Prepared By: Padmavathi Bindulal
Assistant Professor, CE,CSPIT.

Contents

Introduction

Memory allocation process

Use of functions: malloc ( ), calloc ( ), realloc ( ) and free ( ).

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

- **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 space 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.**

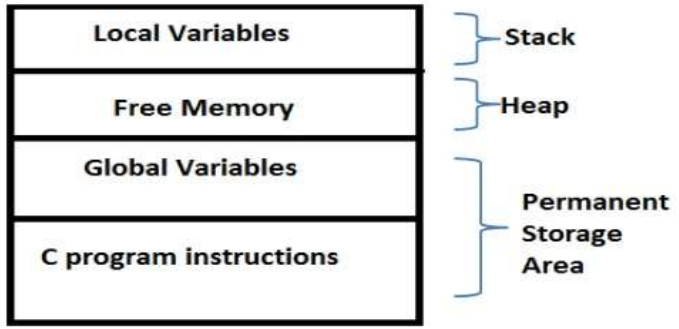
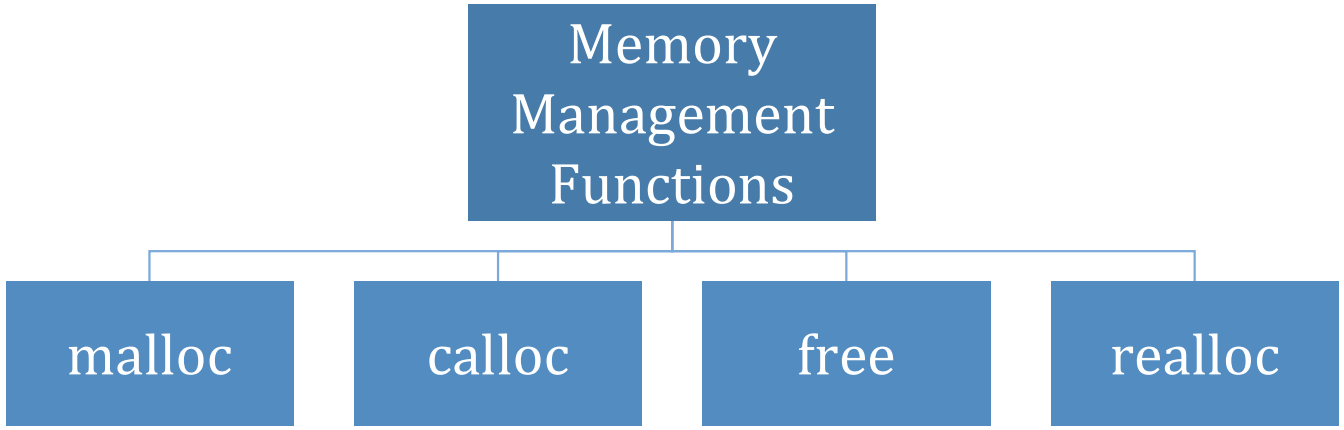


Fig: Storage of a C program

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

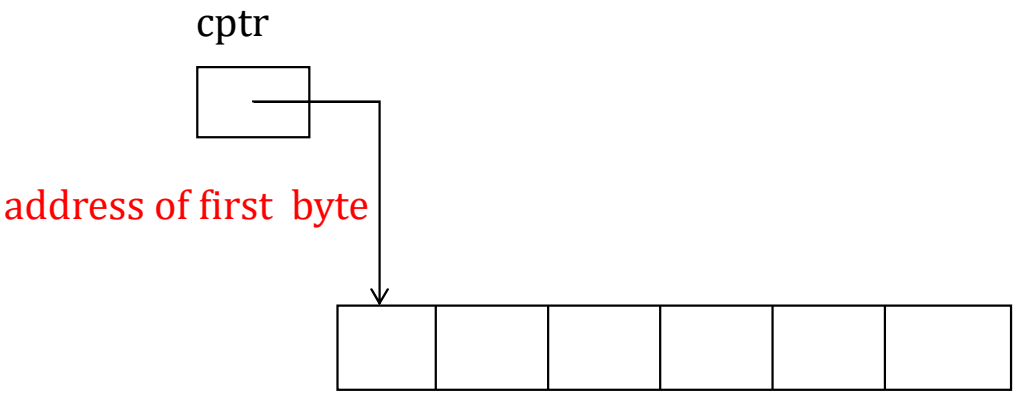
malloc →returns a pointer to an area of memory with size *byte-size*

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()

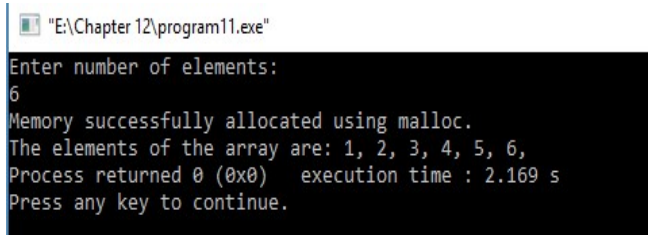
cptr = (char*) malloc(6) , this allocates 6 byte of space for the pointer cptr of type char.



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

Example

```c
//malloc demo
#include <stdio.h>
#include <stdlib.h>
int main()
{
    // This pointer will hold the base address of the block created
    int* ptr;
    int n, i;
    printf("Enter number of elements: \n");  // Get the number of elements for the array
    scanf("%d",&n);
    ptr = (int*)malloc(n * sizeof(int)); // Dynamically allocate memory using malloc()
    // Check if the memory has been successfully allocated by malloc or not
    if (ptr == NULL) {
        printf("Memory not allocated.\n");
        exit(0);
    }
    else {
        // Memory has been successfully allocated
        printf("Memory successfully allocated using malloc.\n");
        // Get the elements of the array
        for (i = 0; i < n; ++i) {
            ptr[i] = i + 1;
        }
        // Print the elements of the array
        printf("The elements of the array are: ");
        for (i = 0; i < n; ++i) {
            printf("%d, ", ptr[i]);
        }
    }
    return 0;
}
```



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.

Example

```c
/*C program to read and print the N student
details using structure and Dynamic Memory Allocation.*/
#include <stdio.h>
#include <stdlib.h>
/*structure declaration*/
struct student
{
    char name[30];
    int roll;
    float perc;
};
int main()
{
    struct student *pstd;
    int n,i;
    printf("Enter total number of elements: ");
    scanf("%d",&n);
    /*Allocate memory dynamically for n objects*/
    pstd=(struct student*)malloc(n*sizeof(struct student));
    if(pstd==NULL)
    {
        printf("Insufficient Memory, Exiting... \n");
        return 0;
    }
    /*read and print details*/
    for(i=0; i<n; i++)
    {
        printf("\nEnter detail of student [%3d]:\n",i+1);
        printf("Enter name: ");
        scanf(" "); /*clear input buffer*/
        gets((pstd+i)->name);
        printf("Enter roll number: ");
        scanf("%d",&(pstd+i)->roll);
        printf("Enter percentage: ");
        scanf("%f",&(pstd+i)->perc);
    }
    printf("\nEntered details are:\n");
    for(i=0; i<n; i++)
    {
        printf("%30s \t %5d \t %.2f\n",(pstd+i)->name,(pstd+i)->roll,(pstd+i)->perc);
    }
    return 0;
}
```

"E:\Chapter 12\program12.exe"

```
Enter total number of elements: 3

Enter detail of student [  1]:
Enter name: aaa
Enter roll number: 23
Enter percentage: 56

Enter detail of student [  2]:
Enter name: bbb
Enter roll number: 36
Enter percentage: 65

Enter detail of student [  3]:
Enter name: ccc
Enter roll number: 96
Enter percentage: 78

Entered details are:
                          aaa      23   56.00
                          bbb      36   65.00
                          ccc      96   78.00

Process returned 0 (0x0)   execution time : 30.298 s
Press any key to continue.
```

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.

Calloc()

```
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

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

The new memory block may or may not begin at the same place as the old one.

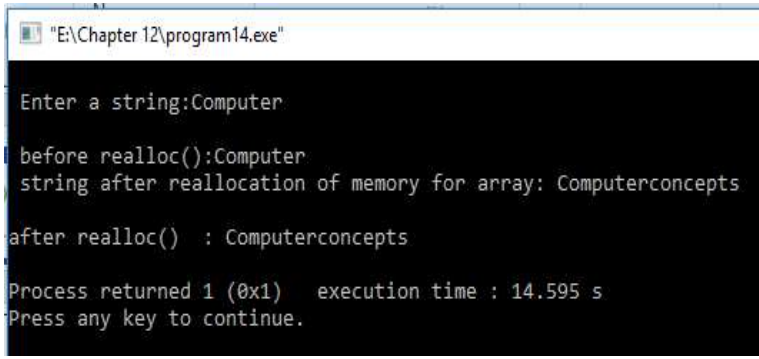
If it fails in locating additional space, it returns NULL pointer and the original block is lost.

Example

Write a program using a character string in a block of memory space created by **calloc ()** and then modify the same to store a larger string using **realloc ()** function. **(Dynamic Array)**.

Example

```c
//realloc demo
#include<stdio.h>
 main()
{
char *a;
a=(char *)calloc(10,sizeof(char)); //allocate 10 bytes of memory
printf("\n Enter a string:");
scanf("%s",a);
printf("\n before realloc():");
printf("%s",a);
a=(char *)realloc(a,20*sizeof(char)); //allocate 20 bytes memory
printf("\n string after reallocation of memory for array: ");
scanf("%s",a);
printf("\nafter realloc()   : ");
puts(a);
free(a);
}
```

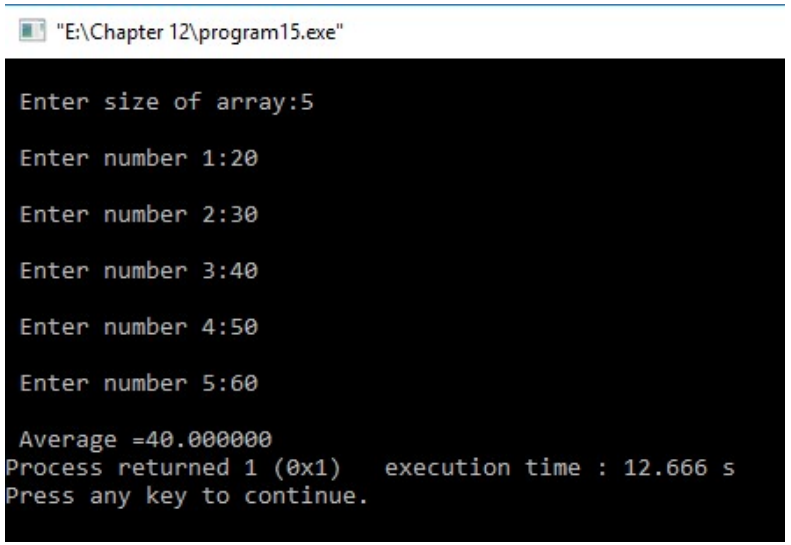


Example

Take the elements of the array using user input, read them and print the sum of all elements along with inputted array elements using Dynamic Memory Allocation.

Example

```c
#include<stdio.h>
 main()
{
int *p;
float *avg;
int i,n;
printf("\n Enter size of array:");
scanf("%d",&n);
p=(int *)calloc(n,sizeof(int));
avg=(float *)malloc(sizeof(float));
*avg=0;
for(i=0;i<n;i++)
{
printf("\n Enter number %d:",i+1);
scanf("%d",p);
*avg=*avg+*p;
p++;
}
*avg=*avg/n;
printf("\n Average =%f",*avg);
free(p);
free(avg);
}
```



Thank You.