Advance C programming

Module II

Static and Dynamic Memory Allocation

Memory allocation:

In programming, it is necessary to store computational data. These data are stored in memory. The memory locations for storing data in computer programming is known as Variables. The variables have a specific datatype.

Memory can be allocated in two ways:

- Static memory allocation
- Dynamic memory allocation

In static memory allocation, once the memory is allocated, the memory size is fixed while in dynamic memory allocation, once the memory is allocated, the memory size can be changed.

Memory allocation

Static Memory allocation | Dynamic memory allocation

Static Memory allocation:

Static memory allocation is also known as compile-time memory allocation because the memory is allocated during compile time. The memory that the program can use is fixed (ie) it cannot allocate or deallocate memory during program's execution.

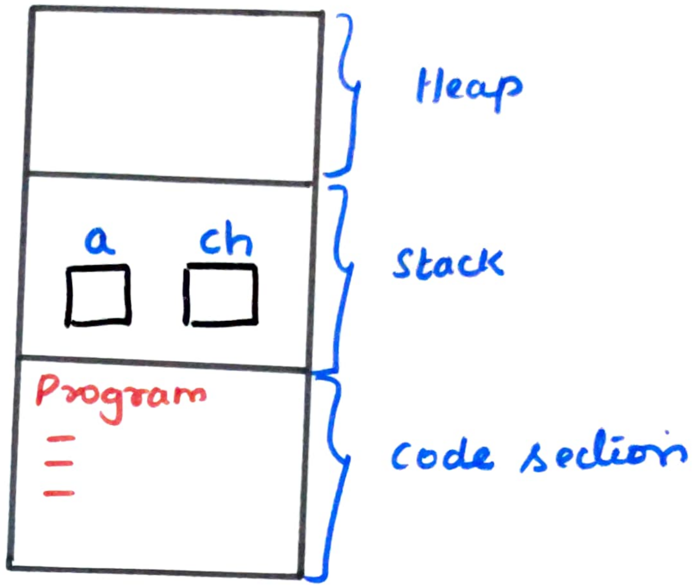
In many applications, it is not possible to predict how much memory will be needed by the program at run time.

Properties:

1. Memory allocation is done during compile time
2. Stack memory is used here
3. Memory cannot be changed while executing a program
4. It is fast and saves running time
5. The allocation process is simple
6. Less efficient compared to dynamic memory allocation.

Example:

```
int main () {
    int a;
    char ch;    }
```

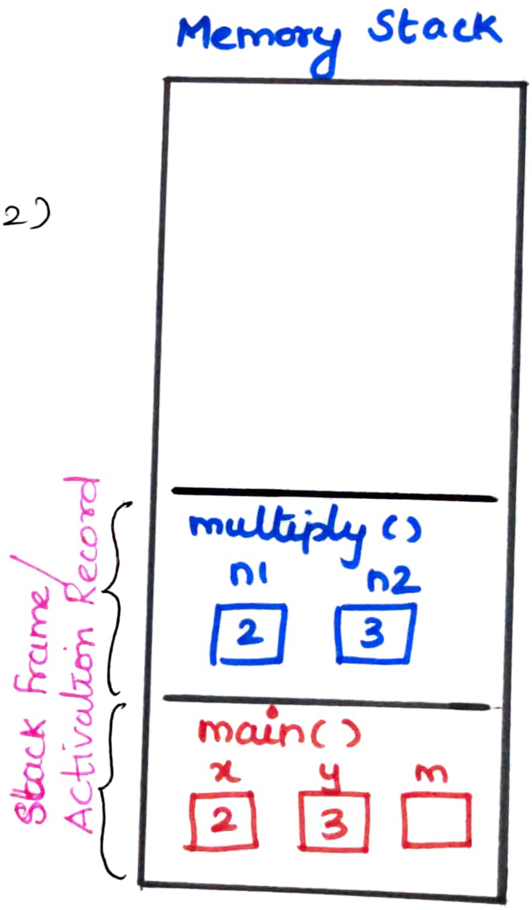


a → 4 bytes

ch → 1 byte

Program :

```
#include<stdio.h>
int Multiply (int n1, int n2)
{
  return n1*n2;
}
int main ()
{
  int x=2;
  int y=3;
  int m = Multiply(x,y);
  printf ("%d", m);
  return 0;
}
```



For every function in the program, the variables will take some part of the stack section which is called as Activation record (or) stack frame. and it will be deleted by the compiler when it is not in use.

Advantages:

* Simple usage
* Allocation and deallocation are done by the compiler
* Efficient execution time
* It uses stack data structures.

Disadvantages:

* Memory wastage problem.
* Exact memory requirements must be known
* Memory can't be resized once after initialization

Dynamic Memory allocation :

Dynamic memory allocation is also known as Runtime memory allocation because the memory is allocated during runtime or program execution. The allocation and release of the memory space can be done using the library functions of stdlib.h header file.

Properties:

1. Memory is allocated at runtime
2. Memory can be allocated and released at any time.
3. Heap memory is used here.
4. Dynamic memory allocation is slow
5. More efficient compared to static memory allocation.
6. The allocation process is complicated
7. Memory can be resized dynamically or reused.

The library functions of the stdlib.h header file, which helps to allocation and deallocation are

1. malloc()
2. calloc()
3. realloc()
4. free()

Malloc ()

The "malloc" or "memory allocation" method in C is used to dynamically allocate a single large block of memory with the specified size. It returns a pointer of type void which can be cast into a pointer of any form.

Syntax:

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

Example:

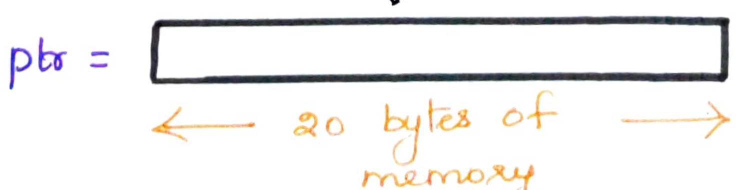
```
ptr = (int*) malloc (100* sizeof (int));
```

Since the size of int is 4 bytes, this statement will allocate 400 bytes of memory. And, the pointer ptr holds the address of the first byte in the allocated memory.

→ 4 bytes

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

↓



If allocation fails due to insufficient space, it returns a NULL pointer.

Example:

```
#include <stdio.h>
# include < stdlib. h>
int main ()
{

    int * ptr ;
    int n, i ;

    printf (" Enter no. of elements : ");
    scanf (" %d", &n);  // Read No. of elements for the array
    printf (" Entered no. of elements : %d \n", n);

    // Dynamic memory allocation using malloc()
    ptr = (int*) malloc (n * sizeof (int));

    // check if memory allocation is successfull
    if( ptr == NULL)
    {
        printf(" Memory not allocated \n");
        exit(0);
    }

    else
    {
     // memory has been successfully allocated
     printf (" Memory allocated successfully using malloc \n");
     for ( i = 0; i < n ; i++)
      {
         ptr[i] = i+1 ;
         printf (" %d  ", ptr[i]); // Prints the elements of the
                                               array
      }
    }
    return 0; }
```

Output:

Enter No. of elements: 5

Memory allocated successfully using malloc.

//The elements of the array are

1 2 3 4 5

Calloc()

The "calloc" or "contiguous memory allocation" method in c is used to dynamically allocate the specified number of blocks of memory of the specified type. It is very much similar to malloc() but has two different points and these are:

1. It initializes each block with a default value '0'.
2. It has two parameters or arguments when compared to malloc().

Syntax:

```
ptr = (cast-type*) calloc(n, element-size);
```

n → no. of elements

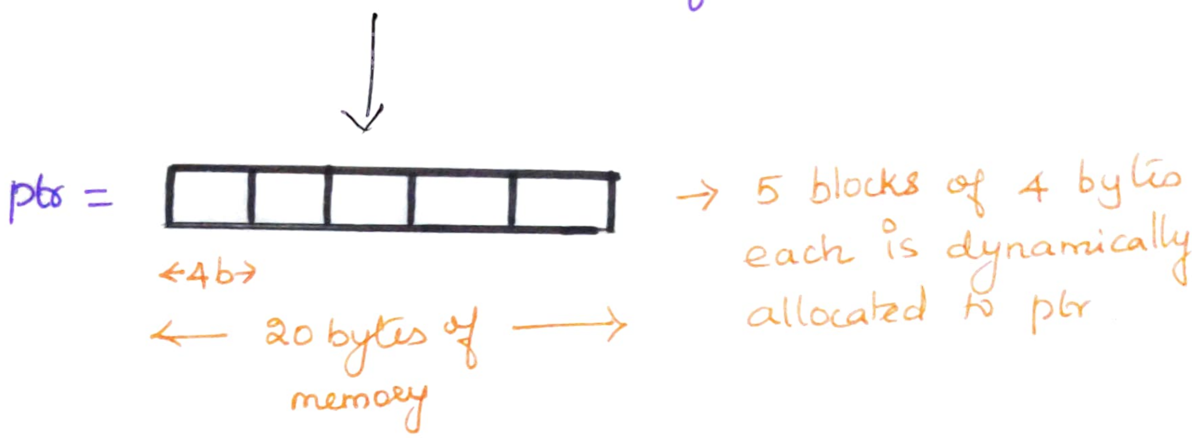
element-size → size of each element.

Example:

```
ptr = (int*) calloc(5, sizeof(int));
```

This statement allocates contiguous space in memory for 5 elements each with the size of int.

```c
int *ptr = (int*) calloc( 5, sizeof(int));   // sizeof(int) → 4 bytes
```



If space is insufficient, allocation fails and returns a **NULL** pointer.

Example:

```c
#include<stdio.h>
# include < stdlib.h>
int main()
{
 int *ptr;
 int n, i;
 n=5;
 printf(" Entered No. of elements : %d \n", n);
 // Dynamic memory allocation using calloc()
 ptr = (int*)calloc(n, sizeof(int));
 // check memory is allocated successfully or not
 if (ptr == NULL)
 {
   printf("Memory not allocated");
   exit(0);
 }
```

```
else
{
  // Display Memory has been successfully allocated
  printf (" Memory successfully allocated using calloc.\n");
  // 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;
}
```

Output:

Entered No. of elements: 5

Memory successfully allocated using calloc.

The elements of the array are:

0 0 0 0 0

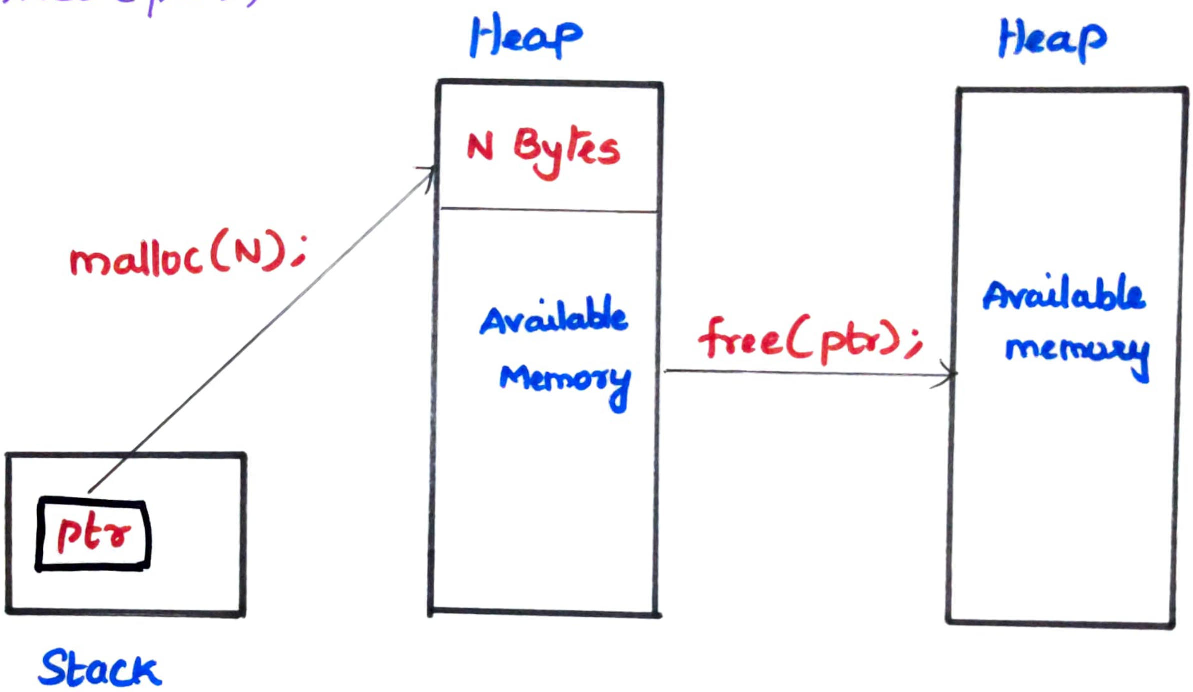
free()

"free" method in C is used to dynamically de-allocate the memory. The memory allocated using functions malloc() and calloc() is not de-allocated on their own.

Hence the free() method is used, whenever the dynamic memory allocation takes place. It helps to reduce wastage of memory by freeing it.

Syntax:

```
free(ptr);
```



Example:

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

int main()
{
int *ptr;
int n;
n=5;
printf("Entered No. of elements: %d\n", n);
```

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

if (ptr == NULL)
{
    printf(" Memory not allocated. \n");
    exit(0);
}
else
{
 printf(" Memory successfully allocated using malloc. \n");

 // freeing the memory
 free(ptr);
 printf(" Malloc memory successfully freed. \n");

 }
 return 0;

}
```

Output:

```
Entered No. of elements : 5
Memory successfully allocated using malloc
Malloc Memory successfully freed.
```

realloc ()

"realloc" or "re-allocation" method in c is used to dynamically change the memory allocation of a previously allocated memory.

re-allocation of memory maintains the already present value and new blocks will be initialized with the default garbage value.

Syntax :

```
ptr = realloc ( ptr , newSize);
```

Example :

4 bytes

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

↓

ptr = 

← 20 bytes →

↓

```
ptr = realloc( ptr, 10* sizeof(int));
```

↓

ptr = 

← 40 bytes of memory →

If space is insufficient, allocation fails and returns a NULL pointer.

Program:

```
#include <stdio.h>
# include <stdlib.h>
int main ()
{
int *ptr, n=5;
printf ("Entered No. of elements: %d \n", n);
ptr = (int *) calloc (n, sizeof(int)); // dynamic allocation
                                         using calloc
if (ptr == NULL)
{
   printf ("Memory not allocated. \n");
   exit (0);
}
else
{
  printf ("The elements of the array are: ");
  for (i=0; i<n; i++)
      printf ("%d", ptr[i]);

  n = 10;

  ptr = realloc (ptr, n*sizeof(int)); //reallocation

  printf ("The elements of the array are: ");

  for (i=0; i<n; i++)
      printf (" %d ", ptr[i]);
  free (ptr);   // freeing the memory
}
return 0;
}
```

Output:

Entered No. of elements: 5

The elements of the array are: 0 0 0 0 0

// after reallocation

The elements of the array are:

0 0 0 0 0 0 0 0 0 0

Advantages:

1. This allocation method has no memory wastage
2. The memory allocation is done at run time
3. Memory size can be change based on requirements during run time
4. If memory is not required, it can be freed.

Disadvantages:

1. It requires more execution time due to execution during runtime
2. The compiler does not help with allocation and deallocation.