ShreeSoft INFORMATICS

Memory Management in C

Introduction:

Memory management can be defined as the process to manage computers memory. Memory management is required to avoid shortage or wastage of memory and to make sure allocation takes place efficiently.

During the declaration of arrays, sometimes the exact memory is not known to us, and that is why we declare an array of max size which results in memory wastage or shortage. To avoid such case we can use Dynamic memory allocation.

There are two types of memory allocation in c language

1) **Static memory allocation:** Static Memory is allocated for declared variables by the compiler. The address can be found using the address of operator and can be assigned to a pointer. The memory is allocated during compile time.

2) **Dynamic Memory Allocation:** is manual allocation and freeing of memory according to your programming needs. Dynamic memory is managed and served with pointers that point to the newly allocated memory space in an area which we call the heap.
   Now you can create and destroy an array of elements dynamically at runtime without any problems. To sum up, the automatic memory management uses the stack, and the C Dynamic Memory Allocation uses the heap.
   There are 4 function used for dynamic memory allocation in c language. This all 4 function comes under the header file stdlib.h.

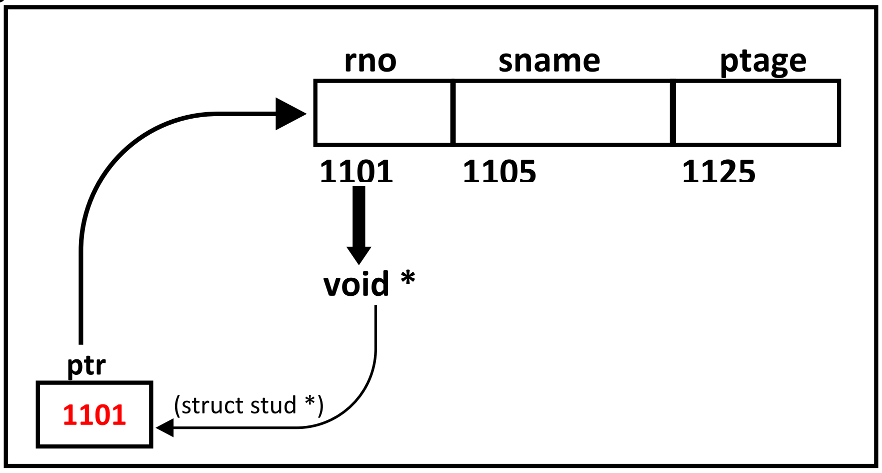
   1) **malloc( ) :** This function used to allocate memory for user defined structured datatype. If this function allocate memory successfully then it return base address of memory block otherwise it return NULL pointer.
      Syntax:

      ```
      void* malloc(size_t  size);
      ```

      Where
      Void* →This is generic pointer. It means that we can store any type (char, int, float, double etc.) of variable address in generic pointer
      size_t → It represents unsigned int.

      Example:

      ```
      struct stud
      {
          int rno;
          char sname[20];
          float ptage;
      };
      struct stud *p;
      ```



      ```
      p=(struct stud *) malloc(sizeof(struct stud));
      ```

2) **calloc( ) :** This is Contiguous memory allocation. This function used to allocate memory for array.
   Syntax: p = (cast-type*)calloc(n, element-size);
   Example Program:

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
   int *p;
   int n, i;
   printf("Enter number of elements:");
   scanf("%d",&n);
   // Dynamically allocate memory using calloc()
   p = (int*)calloc(n,sizeof(int));
   if (p==NULL) {
      printf("\n Memory allocation Failed");
   }
   else {
      for(i=0;i<n;i++) {
         printf("\n Enter Number:");
         scanf("%d",&p[i]);
      }
      printf("\n The elements of the array are:");
      for (i=0;i<n;i++)  {
         printf("%4d",p[i]);
      }
   }
}
```

3) **realloc( ):** This function used to increase or decrease the size of array.
   Syntax: p = realloc(p, newSize);

```
#include<stdio.h>
#include <stdlib.h>
int main()
{
   int *p;
   int n, i;
   char ch;
   printf("Enter number of elements:");
   scanf("%d",&n);
   // Dynamically allocate memory using malloc()
   p = (int*)calloc(n, sizeof(int));
   if (p==NULL)
   {
      printf("\n Memory allocation Failed");
   }
   else
  {
      for(i=0;i<n;i++)
      {
         printf("\n Enter Number:");
         scanf("%d",&p[i]);
      }
      printf("\n The elements of the array are:");
      for (i=0;i<n;i++)
      {
         printf("%4d",p[i]);
      }
      printf("\n Do you want to add more element[Y/N]:");
      fflush(stdin);
      ch=getchar();
      if(ch=='Y'||ch=='y')
      {
           printf("\n Enter how many element to be add more:");
           scanf("%d",&n);
           p = realloc(p,n*sizeof(int));
           for(i=5;i<n;i++)
           {
              printf("\n Enter Number:");
              scanf("%d",&p[i]);
           }
           printf("\n The elements of the array are:");
           for (i=0;i<n;i++)
           {
              printf("%4d",p[i]);
           }
        }
    }
}
```

4) Free( ) : This function used free or release memory.