Dynamic memory allocation (DMA) in C Programming

Using array in programming, we allocate a fixed size for our data. This size can't be increased or decreased while execution of the program. We can't change it even if the size allocated is more or less than our requirement. This type of allocation of memory is called **Static Memory Allocation**. This leads to wastage or shortage of memory.

Fortunately, C allows programmer to allocate memory dynamically i.e. during run time and this process is called dynamic memory allocation. By allocating memory dynamically, we can use only the amount of memory required for us.

For this, C has four built in functions under **"stdlib.h"** header files for allocating memory dynamically. They are:

- malloc()
- calloc()
- realloc()
- free()

malloc()

It allocates a requested size of memory. The memory allocated by **malloc()** are not initialized and hence contains garbage value. It returns a pointer of void type which can be casted into any form.

Syntax of malloc

```
ptr = (cast_type*)malloc(SIZE);
```

For example,

```
ptr = (int*)malloc(n*sizeof(int)); // value of n can be provided in run time
```

Example #1: C program to sort number in ascending order by using malloc function. Use free to release memory.

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int i,j,temp,n;
    int *p;
    printf("Enter value of n: ");
    scanf("%d",&n);
```

```
    p=(int*)malloc(n*sizeof(int));
    printf("Enter values\n");
    for(i=0;i<n;i++)
       scanf("%d",&p[i]);
    for(i=0;i<n;i++)
    {
       for(j=i+1;j<n;j++)
       {
          if(p[i]>p[j])
          {
             temp=p[i];
             p[i]=p[j];
             p[j]=temp;
          }
       }
    }
    printf("Ascending order\n");
    for(i=0;i<n;i++)
       printf("%d\n",p[i]);
    free(p);
    return 0;
}
```

In this program, memory required is allocated at run time. *malloc* function is used to allocate memory. The value of *n* is entered by user and *n* numbers are also entered. Then these numbers are sorted using bubble sort algorithm. The allocation of memory can be changed by changing the value of *n***.** Hence, we can use desired bytes of memory. At last, *free()* releases the used memory.

Output

```
Enter value of n: 5
Enter values
11
31
-2
5
17
Ascending order
-2
5
11
17
31
```

calloc()

It also allocates a requested size of memory like **malloc().** But the memory allocated by calloc is divided into small equal sizes while memory allocated by malloc is not divided. The memory allocated by calloc is initialized to zero.

Syntax of calloc

```
ptr = (cast_type*)calloc(n,element_size);
```

For e.g.

```
ptr = (float*)calloc(n,sizeof(float));
```

Example #2: C program to find squares by using calloc function. Use free to release memory.

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
   int i,n;
   int *p;
   printf("Enter value of n: ");
   scanf("%d",&n);
   p=(int*)calloc(n,sizeof(int));
   printf("Enter values\n");
   for(i=0;i<n;i++)
      scanf("%d",&p[i]);
   for(i=0;i<n;i++)
      printf("Square of %d = %d\n",p[i],p[i]*p[i]);
   free(p);
   return 0;
}
```

Here, the memory is allocated in run time by using **calloc** function. It divides the total memory allocated into small equal fragments as specified by the user. In this program, n fragments each of size equal to *sizeof(int)* is allocated. The value of *n* is entered by user. Then, n integers are entered and their square are printed. At last, *free()* releases the used memory.

Output

```
Enter value of n: 5
Enter values
-4
13
2
7
11
Square of -4 = 16
Square of 13 = 169
Square of 2 = 4
Square of 7 = 49
Square of 11 = 121
```

realloc()

As the name suggests, it is used to reallocate the memory previously allocated by using malloc or calloc in case the memory is excess or insufficient.

Syntax of realloc

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

For e.g.

```
ptr = realloc(ptr,100);
```

Example #3: C program to use realloc() to reallocate memory.

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
   int i,newsize,size;
   int *p;
   printf("Enter size of list: ");
   scanf("%d",&size);
   p=(int*)malloc(size*sizeof(int));
   printf("Enter %d numbers\n",size);
   for(i=0;i<size;i++)
      scanf("%d",&p[i]);
   printf("The numbers in the list are\n");
   for(i=0;i<size;i++)
      printf("%d\n",p[i]);
   printf("Enter new size of list: ");
   scanf("%d",&newsize);
   p=realloc(p,newsize*sizeof(int));
   if(newsize>size)
   {
      printf("Enter %d numbers\n",newsize-size);
      for(i=size;i<newsize;i++)
         scanf("%d",&p[i]);
   }
   printf("The numbers in the list are\n");
   for(i=0;i<newsize;i++)
      printf("%d\n",p[i]);
   return 0;
}
```

In this program, **realloc()** is used to change the size of the memory allocated. Here, an initial size of list is entered by user. The numbers are stored and displayed. Again a new size is entered by user. If the new size is greater than old size then additional numbers are entered by user. Finally the numbers are printed again.

Output

```
Enter size of list: 3
Enter 3 numbers
23
10
7
The numbers in the list are
23
10
7
Enter new size of list: 5
Enter 2 numbers
2
9
The numbers in the list are
23
10
7
2
9
```

free()

This function is used to release the memory if it is of no use anymore.

Syntax of free

```
free(ptr);
```

Advantages of using DMA

- It prevents overflow and underflow of memory. Sometimes, the memory allocated may be more and sometimes it may be less. DMA allows us to shrink or expand the allocated memory. Hence, there will be no overflow or underflow.
- Programmer doesn't need to know about required memory space. So it prevents unnecessary updating of program.

Disadvantage of using DMA

- Program often becomes long and complex. Using array is much ore simpler and easier than using functions like malloc, calloc, realloc and free.
- Memory fragmentation: Sometimes it may be a case that we have sufficient memory as required but they can't be used because they are fragmented.
- User is responsible for freeing up memory when finished with it.

Submitted by Sagun Shrestha