**Dinamik Bellek Yönetimi(Dynamic Memory Allocation)**

C dilinde 3 farklı tip nesne ömrü vardır. Bunlar:

1-) Otomatik Ömürlü Nesneler (Automatic Lifetime Objects) : Temelde bilinmesi gereken en önemli durum bu nesnelerin stack bellek alanında yaratılıp ömürleri bittikten sonra(Faaliyet alanının dışına çıkıldığında) bu bellek alanında yok edilirler. Otomatik ömürlü nesneler ikiye ayrılır. Bunlar:

a-) Yerel Nesneler (Local Objects) : Bu nesneler fonksiyon çağrısında akış bu nesnelerin bildirimlerine geldiği zaman hayata gelirler, fonksiyon çağrısı bittiğinde yok edilirler(stack bellekten atılırlar). Örneğin :

int main(void)

{

int x; //yerel değişken

}

b-) Fonksiyonların Parametre Değişkenleri(Function’s parameter variables) : Bu nesneler yerel nesnelerin fonksiyonların en başında bildirilmesine benzetilebilir. Tahmin edilebileceği gibi bu nesneler de benzer olarak fonksiyon çağrısı bittiği zaman yok edilmektedir.

void foo(int x,int y); //parametre değişkenleri

2-) Statik Ömürlü Nesneler (Static Lifetime Objects): Statik ömürlü nesneler 3 gruba ayrılmaktadır. Bunlar:

a-) statik yerel nesneler (static local objects): Bu değişkenler fonksiyonlar içerisinde bildirilen değişkenlerdir. Yerel değişkenlerden en önemli farkları fonksiyon çağrısında akış onların bildirimine geldiğinde hayata gelip program sonuna kadar yaşamalarıdır. Ancak yerel değişkenler fonksiyon çağrısı bittiğinden yok edilirler ve fonksiyon tekrar çağrılınca yeniden hayata gelirler. Aşağıdaki örnekte normal değişken ile statik yerel değişken arası fark anlatılmak istenmiştir:

#include <stdio.h>

void func(void)

{

static int x = 10;

int y = 10;

printf(“x : %d\n”,++x);

printf(“y : %d\n”,++y);

}

int main(void)

{

func();

func();

func();

}

b-) statik global nesneler (static global objects): Bu nesneler program başladığı zaman yaratılır. Program bitene kadar hayatta kalırlar. Statik yerel nesnelerden farkı fonksiyon çağrılmasına gerek olmadan hayata gelmeleridir diyebiliriz. Bu nesnelere ilk değer verilmediği takdirde default olarak sıfır değeri alırlar. İlk değer verilmemiş statik global değişkenler BSS bellek alanında tutulur. Eğer ilk değer verilmişse bu durumda Data bellek bloğunu kullanmaktadır. Statik global değişkenler aşağıdaki örnekte olduğu gibi tanımlanabilir:  
static int x = 10;//Data bellek bloğunda tutulur.

static int y; // BSS bellek bloğunda tutulur.

c-) global nesneler (global objects): Bu nesneler hem tutulduğu bellek bloğu açısından hem faaliyet alanı açısından statik global değişkenlerle oldukça benzer özellik gösterirler. Aralarındaki en önemli fark bu değişkenlerin external linkage’a sahip olabilmesidir. Oysa statik global değişkenler internal linkage’a aittir. Bu da şimdiki bilgilerimize göre dışardan erişime kapalı anlamına gelir. İlerde ele alacağımız extern anahtar sözcüğü kullanılarak global değişkenlere başka modüllerden erişilebilir. Ancak statik global değişkenler yalnızca kendi bildirildikleri dosya içerisinde anlamlıdır. Bu açıklamalara göre aşağıdaki örnekleri inceleyiniz:

static int sx; //dışardan erişime kapalı

int gx = 10; //dışardan erişime açık ve Data bellek bloğundan //tutuluyor.

3-) Dinamik Ömürlü Nesneler (Dynamic Lifetime Objects): Dinamik ömürlü nesneler bellek bloğu olarak heap belleği kullanır. Bu bellek alanını kullanmak için özellikle standart kütüphanenin stdlib.h başlık dosyasını eklemek gerekmektedir. Bu kütüphanede önceden rastgele sayı üretiminde de kullandığımız rand() ve srand() fonksiyonları da bulunmaktadır. Dinamik bellek yönetimi için kullanılan ve heap bellek alanını tahsis eden, tahsis edilen alanı büyütüp, küçülten ve bu alanı geri veren fonksiyonlar sırasıyla şu şekildedir:

1-) void\* malloc(size\_t size) fonksiyonu : void\* türünden geri dönüş değerine sahip ve tahsis edilecek toplam bellek byte’ını argüman olarak alan bir fonksiyondur. Bu fonksiyon örneğin 10 elemanlı bir dizi oluşturmak istenirse aşağıdaki gibi kullanılabilir:

#include <stdlib.h>

int main(void)

{

int size;

printf(“Enter size?\n”);

scanf(“%d”,&size);

int\*arr = (int\*)malloc(size \* sizeof(int));

if(arr == NULL)

{

printf(“Bellek yetersiz\n”);

exit(EXIT\_FAILURE);

}

}

şekilde yazılabilir.

**Anahtar Notlar :** Eğer bellek yetersiz ise bu durumda malloc, calloc ve realloc gibi fonksiyonlar NULL pointer döndürün. Belleğin yetersiz olduğu durumlar için tanımsız davranışları önlemek için bu aşamada program sonlandırılmalıdır.

Malloc fonksiyonun geri dönüş değeri daha önceden de belirttiğimiz gibi void\* türünden bir geri dönüş değerine sahiptir. void\* türünden int\* türüne C dilinde doğrudan dönüşüm(standard conversion) geçerlidir. Ancak yine de bu C++’ta geçersiz olduğundan genellikle burda type conversion(tür dönüştürme) operatörü kullanımı kullanımı tavsiye edilir. Malloc ile ilgili diğer bilinmesi gereken diğer bir özellik dizinin elemanları default olarak çöp değer(garbage value) ile hayata başlamıştır. Eğer bellek yetersiz ise bu fonksiyon NULL geri döndürür. Programcı mutlaka belleğin yetersiz olması durumunu da hesaba katmalıdır.

2-) void\* calloc(size\_t \_NumOfElements, size\_t \_SizeOfElements) fonksiyonu : Bu fonksiyon da C’de bellek tahsisatı için kullanılır. İki tane parametre değişkenine sahiptir. Bu parametrelerden birincisi dizinin boyutu, ikinci parametresi ise bir elemanın byte olarak sizeof değerini alır. Malloc fonksiyonunda 2 farklılığı vardır :

a-) Malloc ile oluşturulan dizinin elemanları default olarak çöp değerle hayatına başlar. Diğer taraftan calloc fonksiyonu default olarak elemanları sıfırla hayata başlatır.

b-) Malloc fonksiyonu tek parametre alırken calloc fonksiyon iki parametre alır.

Bu açıklamalara göre calloc ile verilen aşağıdaki örneği inceleyiniz:

#include <stdlib.h>

int main(void)

{

int size;

printf(“Enter size?\n”);

scanf(“%d”,&size);

int\*arr = (int\*)calloc(size, sizeof(int));

if(arr == NULL)

{

printf(“Bellek yetersiz\n”);

exit(EXIT\_FAILURE);

}

}

Temelde mallocla calloc arasında pek de fark olmadığını söyleyebiliriz.

3-) void\* realloc(void \*\_Memory, size\_t \_NewSize) fonksiyonu : Geri dönüş değeri void\* olan, parametre değişkenlerinin ilki daha önceden malloc veya calloc ile dinamik olarak tahsis edilmiş olan dizinin kendisini, ikinci parametre olarak ise son durumda toplam size değerini alan, dinamik olarak tahsis edilmek bellek bloğunu büyütmek veya küçültmek için kullanılan bir fonksiyondur. Bu fonksiyonun her daim dinamik olarak tahsis edilmiş bir bellek bloğunu büyütmek veya küçültmek için kullanılmalıdır. Realloc fonksiyonunun kullanımıyla ilgili aşağıdaki örneği inceleyiniz :

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#define randomize() srand(time(NULL))

void printArray(int arr[],int size)

{

for(int i = 0;i < size;++i)

{

printf("%d ",arr[i]);

}

printf("\n");

}

void setRandomArray(int arr[],int size)

{

for(int i = 0;i < size;++i)

{

arr[i] = rand() % 100;

}

}

int main(void)

{

randomize();

int size;

printf("Enter Size?\n");

scanf("%d",&size);

int\* arr = malloc(size \* sizeof(int));

if(arr == NULL)

{

printf(“Bellek yetersiz\n”);

exit(EXIT\_FAILURE);

}

setRandomArray(arr,size);

arr = realloc(arr,size + 10);

printArray(arr,size + 10);

}

4-) void free(void \*\_Memory) fonksiyonu : free fonksiyonu, dinamik olarak tahsis edilmiş bellek alanını boşaltmak için kullanılan oldukça önemli bir standart kütüphane fonksiyonudur. Bu fonksiyon çağrılmadığı takdirde programda memory leak oluşacaktır. Bu da programda çökmelere sebep olabilir. Parametre olarak serbest bırakılacak bellek bloğunun kendisini alır.

Bellek alanı free ile serbest bırakıldıktan sonra son durum dangling hale gelecektir. Yani NULL pointer’a eşit olacaktır. Aşağıdaki örnekte free fonksiyonu gösterilmiştir:

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#define randomize() srand(time(NULL))

void printArray(int arr[],int size)

{

for(int i = 0;i < size;++i)

{

printf("%d ",arr[i]);

}

printf("\n");

}

void setRandomArray(int arr[],int size)

{

for(int i = 0;i < size;++i)

{

arr[i] = rand() % 100;

}

}

int main(void)

{

randomize();

int size;

printf("Enter Size?\n");

scanf("%d",&size);

int\* arr = malloc(size \* sizeof(int));

if(arr == NULL)

{

printf(“Bellek yetersiz\n”);

exit(EXIT\_FAILURE);

}

setRandomArray(arr,size);

arr = realloc(arr,size + 10);

if(arr == NULL)

{

printf(“Bellek yetersiz\n”);

exit(EXIT\_FAILURE);

}

printArray(arr,size + 10);

free(arr);

}