c programlama dilinde linked list (bağlı listeler)

Bir zaman önce C dilinde yapılar ve kullanımları (structures in C) başlıklı bir yazı yazmıştım. Eğer struct konusuna hakim değilseniz bir göz gezdirmenizi öneririm.

Bu yazımda, C programlama dilindeki linked listlerin ne olduğundan, neden ihtiyaç duyduğumuzdan bahsedeceğim. Ardından da kod parçacıkları ile örnek vereceğim.

Linked List nedir?

Düğüm (node) adı verilen ve göstericiler (pointer) sayesinde birbirine bağlanan **kendine dönüşlü yapıların doğrusal** bir şeklidir.

Bu noktada **kendine dönüşlü yapı**nın ne demek olduğuna örnek vermek gerekirse;

```
    struct Node{
        int data;
        struct Node *next;
    }
```

Buradaki **Node** ismindeki struct, içerisinde data isminde int türünde bir değişken ve kendi tipinde bir struct gösteren pointer'a sahiptir. Buna **kendine dönüşlü yapı** denmektedir. Bu tanım şu an hafızamızda pek yer etmemiş olabilir, yazının devamında neden böyle birşey yaptığımızı daha iyi anlayabiliriz.

Linked list tanımını yaparken bir de doğrusal diye bir terim kullandık. Bunun anlamı esasında fiziksel olarak değildir. Hafızada, her node farklı bir yerde olur, ancak biz çalışmamız sırasında işlerimizi rahat ilerletebilmek için buna doğrusal deriz. Çünkü kağıda kalemle çizdiğimizde, linked listlerde her node birbirinin ardından gelir. Yazının devamında göreceğiz.

Bir linked liste, listin ilk düğümünü gösteren gösterici sayesinde erişilir. Ve genellikle son düğümdeki pointer, listin sonu olduğunu belirtmek amacıyla **NULL** değere eşitlenir.

Linked listlerde veriler dinamik olarak tutulur. Yani her düğüm ihtiyaç olduğunda silinir veya oluşturulur. Yukarıda kendine dönüşlü yapılardan bahsetmiştim. Bir linked list içerisinde sadece kendi tipinden değil, başka türden structlar da olabilir. Yani **sadece kendi tipinden structlar olacak diye bir kural yoktur.**

Linked list oluştururken ilk olarak yapmamız gereken malloc ve sizeof işlemlerini kullanmaktır. sizeof ile elimizdeki structın boyutunu alırız, ve malloc fonksiyonu ile hafızada o kadar alanlık bir yer ayırırız.

Linked listlere yeni bir düğüm ekleme işlemini 3 farklı yere yapabiliriz. Listenin en başına, listenin en sonuna, ve herhangi iki düğüm arasına.

Şöyle bir yapı oluşturduğumuzu düşünelim;

```
typedef struct linked_list {
    int data;
    struct linked_list *next;
} NODE;
```

Bu şekilde bir yapıya art arda yeni düğümler eklemek için aşağıdaki kod parçacığını kullanabiliriz.

```
void insert(int value){

    if(head==NULL){ /// ilk düğüm oluşturulmadıysa önce onu oluşturuyoruz.
        head = (NODE *)malloc(sizeof(NODE));
        head->data = value;
        head->next = NULL;
        tail = head; /// Sadece ilk düğüm için, head = tail
    } else {
        NODE *temp = (NODE *)malloc(sizeof(NODE)); /// Eklenecek düğümler için memory allocation
        temp->data = value;
        temp->next = NULL;
        tail->next = temp;
        tail = tail->next;
    }
    printf("Eklenen data %d\n", value);
}
```

Listeye başından ekleme yapmak için şu kod parçacığını kullanabiliriz;

```
void head_insert(int value){

    if(head==NULL){ /// İlk düğüm oluşturulmadıysa..
        head = (LL *)malloc(sizeof(NODE));
        head->data = value;
        head->next = NULL;
        tail = head; /// Sadece ilk düğüm için, head = tail
    } else {
        NODE *temp = (NODE *)malloc(sizeof(NODE)); /// Eklenecek düğümler için memory allocation
        temp->data = value;
        temp->next = head;
        head = temp; /// Listenin başına yeni düğüm ekledik.
    }
    printf("Listenin başına eklendi --> %d\n", value);
}
```

Herhangi bir elemandan önce veya sonra ekleme yapmayı da aynı yukarıdaki mantıkla yapabiliriz. while döngüsü ile listeyi dolaşırız. Oluşturduğumuz yeni düğümü veri kaybı olmadan iki düğüm arasına yerleştiririz.

```
void insert_after(int insert, int foo){ ///insert değerini foo değerinden sonrasına ekliyoruz
    LL *curr, *temp; /// gerekli işlemleri yapabilmek için 2 pointera ihtiyacımız var
    if(head!=NULL){ /// tabi ki bu işlemi liste boş değilse yapabiliriz

        curr = head; /// listenin en başından başlayacağız.

        while(curr!=NULL){ /// next değeri NULL olan elemana kadar listeyi dolaşıyoruz (yani listenin so
            if(curr->data==foo){ /// aradığımızı bulduk

                temp = (NODE *)malloc(sizeof(NODE));
                temp->data = insert;
                temp->next = curr->next;
                if(curr->next==NULL){
                    tail = temp;
                }
                curr->next = temp;
                curr = curr->next;
            }
            curr = curr->next; /// aradığımızı bulamadıysak bulunduğumuz düğümden bir sonrakine geçiyoru
        }
    }
}
```

Daha sonraki yazılarımda linked listler ile ilgili farklı örnekler de vereceğim.

Posted by Gökhan Tunçkale / JANUARY 12, 2013 / 4 Comments / Tagged with bağlı listeler, c, c programlama, data, iterator, linked list, malloc, node, sizeof, struct, structure / Posted in Software, C, Software