2.11.2023

DATA STRUCTERS LAB COURSE – STACK

HW

1- Bir txt dosya içerisindeki kodda var olan parantez hatalarını tespit edecek programı stack array implementation ile yazınız.
Metin dosyasında yazılı kodu açılacak, dosya göstericisi tek tek okuyacak ve bir stağa aktaracaktır yalnızca parantez hata kontrolü yapılacaktır.

Örnek dosya içeriği:

```
int main(){
    while(1){
            float a = [((4*8) + 7) / 5 ];
            for(int i=0; i<=10; i++){
                    printf("%d\n",i);
            }
    }
}
}
```

*** metin dosyası çalıştırılan c dosyası ile aynı dizinde olmalıdır, eğer değilse metin dosyasının yolu okunmak için fonksiyona verilmelidir.

1- 1-Write a program with a stack array implementation that will detect parentheses errors in the code in a text file.
The code written in the text file will be opened, the file pointer will read it one by one and transfer it to a store, only the parentheses will be checked for errors.

Örnek dosya içeriği:

```
int main(){
    while(1){
            float a = [((4*8) + 7) / 5 ];
            for(int i=0; i<=10; i++){
                    printf("%d\n",i);
            }
    }
}
}
```

*** The text file must be in the same directory as the C file being run, if not, the path to the text file must be given to the function to be read.

send it to **datastr23@gmail.com**

Everyone should upload the code they have developed to their **Github** account in a folder with the assignment questions, with Turkish or Englısh explanations in the required lines, and add **the github account links** to the above e-mail address with the assignment.

TÜRKÇE

Kıymetli dostlarım, dersimiz veri yapıları bu sebeple, verilerin depolanması işlenmesi ve üzerinde değişiklikler yapılmasını mümkün kılan kodlar yazmaya çalışacağız. Bu dersi çokça araştırmak, örnek soru ve sorunlar bulmak ve çözmek gayretinde olmak zorundayız. Bu dersi sevmek ve gayret etmek zorundayız. Temel bir derstir, her teori dersinde yalnızca izlemeyin, bilgisayarlarınızı alıp derse katılın ve tahtaya yazılan ne varsa koda dökün. Mümkün olduğunca aynı gün tekrar etmeye önem verin. Başarı ancak gayretimiz ile gelecektir. Her uygulama dersine kendi bilgisayarlarınız ile gelin. Dijital veya defter notlarınız mutlaka yanınızda olsun. Bir Github hesabını kendiniz için açın. Her hafta birşeyler yüklemeye gayret edin.

Listelerde de, yığın yapısında da, kuyruk yapısında, ağaç yapısında hep bir yapı oluşturmak tanımlamak gerekecektir. Elimiz buna alışmalı. Yığın yapısında teorik olarak oturmayan konuları yeniden dinlemek ve kendinize fayda sağlamak için bu video serisini izleyin: https://www.youtube.com/watch?v=vTI7CuYCtds

Aynı kodları defalarca yazın.

Postfix işlemi için bir örnek senaryo işte burada. Her zaman yaptığımız gibi bir yapı oluşturmakla işe başlayalım. Hemen ardından, initialize fonksiyonunu create etmek gerekir. Yığının top işaretçisi -1'den başlatılmasının nedeni, dizilerde ilk eleman 0. indiste saklanır, eğer dizi ile implemente edilen bir yığın yapısı varsa bu durumda, 0. indise ilk datayı eklemek gerekecekse, onun yerini korumalı ve -1. indis mümkün olmadığından buradan başlatılmalı.

Yığın yapısında verilerin saklanması için mutlaka push()=ekler ve pop()=çıkartır, fonksiyonlarına ihtiyacımız olacaktır.

Dönüş tipi void olan (çünkü tek işi yığına eleman eklemektir) push fonksiyonunu create edelim ve alacağı ilk parametre stack tipinde bir işaretçidir. ikinci parametre ise yığına eklenecek olan data olmalıdır. Bir yığına eleman eklenmek isteniyorsa, yığın tamamen dolu ise bu mümkün olmayacaktır bu nedenle yığının top göstericisi, yığının maksimum kapasitesi kadar var mı kontrol edilmeli. Yığın boyutu 20 ise, diziler 0. indis ile başlandığından en fazla 19. indis hesaba katılır.

Eğer yığın içerisinde eleman alacak yer kaldı ise yığının top göstericisi artırılır ve data yığına eklenir.

Yığından eleman çıkarma işlemine gelecek olursak, burada mühim olan yığın tamamen boş ise çıkaracak eleman yoktur bu nedenle tamamen boş mu kontrolü yapılır. Eğer yığının top göstericisi -1 ise az evvel bahsettik yığında hiç eleman yok demektir. O halde pop() işlemi mümkün değildir.

Aksi durumda ise yığının top göstericisi bir azaltılır ve data silinir.

Herşey başlarken bundan ibaret ancak bitmedi ! Parantez kontrolü yapan ve postfix işlemini gerçekleştiren örnekleri teori derinden hatırlayın, notlarınıza defalarca bakın ve aranızda tartışın.

Bizden istenen bu fonksiyonlar kullanılarak, probleme yönelik program geliştirmek.

Expression kısaltması olan exp adında bir karakter dizisi tanımladık, boyutu 20'dir.

Yine char tipinde bir gösterici tanımladık. Stack tipinde s adında stack değişkeni tanımlandıktan sonra, başlangıç adresi (&s) parametre olarak initialize fonksiyonuna gönderilir. İnitialize fonksiyonunun parametresini hatırlayın stk adında bir gösterici. Buraya kadar pekçok şey tamam, şimdi programda gerçeklenecek olan postfix işlemini kullanıcıdan isteyelim. Bu bir string ifadedir ve exp dizisi içerisine atılır. İfade dizide saklı tutulur.

e adındaki gösterici tanımlandıktan sonra bunun string sonu ifade yani boşluk karalterini gösterinceye kadar dizide gezinmesi sağlanır. Böylece klavyeden yazılın ifade karakter karakter okunur, ta ki okunacak tek bir karakter kalmayıncaya kadar.

Fonksiyona gönderilen parametre değerinin rakam olup olmadığını kontrol eder. Eğer karakter bir rakam ise sıfır olmayan bir değer, aksi takdirde 0 değerini geri döndürür.

Biz kullanıcının klavyeden girdiği ifadedeki rakamları ve operatörleri ayırmak, tespit etmek zorundayız. İfadedeki her rakam öncelikle yığına atılır,eklenir. Bu push fonksiyonu ile mümkündür. Bu göstericinin okuduğu rakamın ancak ascii karşılığıdır, bunu bilidiğimiz sayısal değere dönüştürmek için, göstericinin işaret ettiği rakamdan sıfır çıkartırız. Ve artık bunun bir rakama karşılık geldiğini anladığımızda yığına ekleriz.

Eğer okunan karakter bir rakam değilse operatördür. Çarpma toplama çıkarma gibi işlem yapmaya yarayan artimetik bir operatör. Bu sebeple yığında şimdiye kadar hangi rakamlar saklandı ise çıkartılır (pop()) ve ardından aritmetik işleme tabi tutulur.

Elbette her operatör aynı işlemi yapmıyor, bu nedenle bir swicth case yapısı oluşturup, göstericinin diziden okuduğu karakter ne ise ona uygun işlem yapılmalıdır.

Sonuçta elde edilen son çıktıyı yığınan atar(push()). While döngüsü bitip de okunacak karater kalmadığında, sonucu ekranda görmek için yığından çıkartırız(pop()).

İşte hepsi bu. İyi çalışın, farklı sorular çözün. Sevgiler :)

IN Englısh

Dear friends, our lesson is data structures, so we will try to write codes that make it possible to store, process and make changes to data. We have to research this course a lot, find sample questions and problems, and try to solve them. We have to love this lesson and strive for it. It is a basic course, in every theory lesson, do not just watch, take your computers and join the lesson and put into code everything written on the board. Make it a point to repeat it on the same day as much as possible. Success will only come with our efforts. Come to each practice lesson with your own computers. Be sure to have your digital or notebook notes with you. Open a Github account for yourself. Try to upload something every week.

It will be necessary to define a structure in lists, stack structure, queue structure and tree structure. Our hands must get used to this. Watch this video series to re-listen to topics that do not theoretically fit into the stack structure and benefit yourself: https://www.youtube.com/watch?v=vTI7CuYCtds

Write the same codes over and over again.

Here is an example scenario for the Postfix process. Let's start by creating a structure as we always do. Immediately after, it is necessary to create the initialize function. The reason why the stack is initialized from top pointer -1 is that in arrays, the first element is stored at index 0. If there is a stack structure implemented with the array, in this case, if it is necessary to add the first data to index 0, it should preserve its place and set it to -1. Since index is not possible, it must be started from here.

To store the data in the stack structure, we will definitely need the push()=adds and pop()=subtract functions.

Let's create the push function, whose return type is void (because its only job is to add elements to the stack), and the first parameter it will receive is a pointer of stack type. The second parameter should be the data to be added to the stack. If you want to add an element to a stack, this will not be possible if the stack is completely full, so the top pointer of the stack should be checked to see if it has the maximum capacity of the stack. If the stack size is 20, at most the 19th index is taken into account since the arrays start with the 0th index.

If there is space left in the stack to accommodate elements, the top pointer of the stack is incremented and the data is added to the stack.

Coming to the process of removing elements from the stack, what is important here is that if the stack is completely empty, there is no element to remove, so it is checked whether it is completely empty. If the top pointer of the stack is -1, it means that there are no elements in the stack that we just mentioned. Then pop() operation is not possible.

Otherwise, the top pointer of the stack is decremented by one and the data is deleted.

This is all it started, but it didn't end! Remember the examples of checking parentheses and performing postfix operations deeply in theory, look at your notes many times and discuss them among yourselves.

Developing a problem-oriented program using these functions is requested from us.

We defined a character array called exp, which is short for Expression, and its size is 20.

Again, we defined a pointer of type char. After defining a stack variable named s of stack type, the starting address (&s) is sent to the initialize function as a parameter. Recall that the parameter of the initialize function is a pointer named stk. So far, everything is ok, now let's ask the user for the postfix operation to be performed in the program. This is a string expression and is thrown into the exp array. The expression is reserved in the array.

After the pointer named e is defined, it is allowed to navigate through the array until the end of the string points to the expression, that is, the space character. Thus, the expression typed on the keyboard is read out character by character, until there is not a single character left to read.

It checks whether the parameter value sent to the function is a number. Returns a non-zero value if the character is a number, 0 otherwise.

We have to separate and detect the numbers and operators in the expression that the user enters from the keyboard. Each digit in the expression is first added to the stack. This is possible with the push function. This is just the ascii equivalent of the number read by the pointer. To convert it to the numerical value we know, we subtract zero from the number pointed by the pointer. And now when we realize that it corresponds to a number, we add it to the stack.

If the character read is not a number, it is an operator. An arithmetic operator used to perform operations such as multiplication, addition and subtraction. For this reason, whatever numbers have been stored in the stack so far are popped (pop()) and then subjected to arithmetic processing.

Of course, not every operator performs the same operation, so a switch case structure must be created and the operation must be performed according to the character the pointer reads from the array.

The resulting output is pushed (push()). When the while loop ends and there are no more characters left to read, we pop the result from the stack to see it on the screen (pop()).

That's it. Study well, solve different questions. Best regards.

Arş.Gör.Saliha ÖZGÜNGÖR

```c
#include<stdio.h>
#define STACK_SIZE 20

typedef struct{
    int data[STACK_SIZE];
    int top;
}stack;

void initialize(stack * stk){
    stk->top=-1;
}

void push( stack * stk,int c){
    if(stk->top!= STACK_SIZE-1){
        stk->top++;
        stk->data[stk->top]=c;
    }
    else
        printf("\nStack is Full!!");
}

int pop (stack * stk){
    if(stk->top!=-1){
        return stk->data[stk->top--];
    }
    printf("\n Stack is Empty!!");
    return 0;
}

int main(){
    char exp[20];
    char *e;
    stack s;
    initialize(&s);
```

```
    int n1,n2,n3,num;
    printf("Postfix ifadesini girin: ");
    scanf("%s",exp);
    e = exp;
    while(*e != '\0'){
        if(isdigit(*e)){
            num = *e - 48; //tam say□ya □evirme
            push(&s,num);
        }else{
            n1 = pop(&s);
            n2 = pop(&s);
            switch(*e){
                case '+':
                    n3 = n2 + n1;
                    break;
                case '-':
                    n3 = n2 - n1;
                    break;
                case '*':
                    n3 = n2 * n1;
                    break;
                case '/':
                    n3 = n2 / n1;
                    break;
            }
            push(&s,n3);
        }
        e++;
    }
    printf("\nSonuc =  %d\n",pop(&s));
}
```

Expressıon : 895*+9- ‘\0’

8,9,5,* ---→ 5*9 = 45 push it

8,40+ --→ 40+8 =48 push ı tın stack

48,9 - -→48-9 =39 push ıt agaın ın stack