Tanım

Derleyici tasarımında yığıt sıkça kullanılan bir yapıdır. Aritmetik ifadeleri infix gösteriminden postfix gösterimine çevirmede kullanılırlar. Birçok derleyici aritmetik ifadeyi infix gösteriminden postfix gösterimine dönüştürüp yığıtta saklar. Derleyicilerin ifadeleri postfix gösterimine çevirmelerindeki neden parantezlere olan ihtiyacı ortadan kaldırmaktır.

Infix Gösterim

Operatörlerin işlem yapılacak sayıların arasında olduğu gösterimdir.

Örnek:

(22x10)-8

(11+9)-(25/7)

Postfix Gösterim

Operatörlerin işlem yapılacak ilgili sayıların sonunda olduğu gösterimdir. Parantez kullanımı zorunluluğunu ortadan kaldırır.

Örnek:

22 10 x 8 -

11 9 + 25 7 / -

Operatör Öncelikleri

İlk önceliği parantezler belirler.

Öncelik Sırası:

+, - Sayıların işaretleri

++ , -- Arttırma, azaltma

. * işaretçiler (pointer)

* , / , % Çarpma, bölme, modüler

+ , - Toplama, çıkarma

-= , += , %= , /= , *= Bileşik atama işlemleri

= Atama işlemi

C++ programlama dilinde matematikteki işlem öncelikleri aynen geçerlidir. Öncelikleri eşit olan işlemler soldan sağa doğru hesaplanırlar. Örneği *, / ve % işlemlerinden oluşan aşağıdaki ifade

int sonuc=a*b/2*c%10; ➔ (((a*b)/2)*c)%10 ifadesi şeklinde hesaplanacaktır.

int x=3,y=5;

cout<<x+-y; // Buradaki örnekte ise sayıların işaretlerinin önceliği daha önde olduğu için önce y'nin işareti değiştirilecek sonra x ile toplanacaktır.

Sonuç -2'dir.

Infix-Postfix Dönüşümü

1. Adım: Operatörlerin öncelik sırası belirlenir.

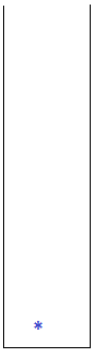
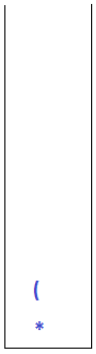
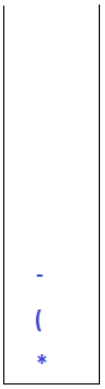
2. Adım: Infix ifadede soldan sağa okuma yapılır ve okunan operatör değil de bir sayı ise yığıta atılmaz postfix (output) ifadeye yerleştirilir.

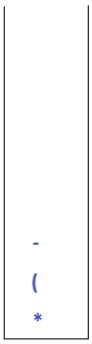
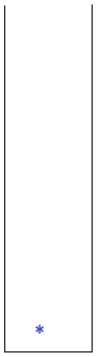
3. Adım: Eğer Infix ifadeden okunan sıradaki eleman operatör ya da parantez ise;

Okunan operatörden daha düşük önceliğe sahip bir operatör gelene kadar yığıttan pop işlemi yap. Sol parantez Infix ifadeden sağ parantez okunana kadar, yığıttan pop edilmez.

Yığıttan pop edilen elemanlar postfix ifadeye yerleştirilir. (Parantezler hariç)

10 * (20 – 8) / 5	Yığıt (Stack) Otuput (Postfix)
*** (20 – 8) / 5**	Yığıt (Stack) Otuput (Postfix) 10

(20 – 8) / 5	Yığıt (Stack) * 	Otuput (Postfix) 10
20 – 8) / 5	Yığıt (Stack) (* 	Otuput (Postfix) 10
– 8) / 5	Yığıt (Stack) (* 	Otuput (Postfix) 10 20
8) / 5	Yığıt (Stack) - (* 	Otuput (Postfix) 10 20

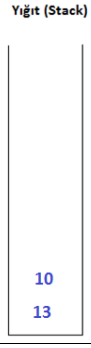
) / 5	Yığıt (Stack) 	Otuput (Postfix) 10 20 8
/ 5	Yığıt (Stack) 	Otuput (Postfix) 10 20 8 -
5	Yığıt (Stack) 	Otuput (Postfix) 10 20 8 - *
	Yığıt (Stack) 	Otuput (Postfix) 10 20 8 - * 5 /

```
string InfixtoPostfix(string infix)
        {
            int uzunluk = infix.length();
            Stack<char>* stack = new Stack<char>();
            string postfix = "";

            for (int i = 0; i < uzunluk; i++)
            {
                if (isdigit(infix[i]))
                {
                    while (isdigit(infix[i])) { postfix += infix[i]; i++; } // Ondalik sayılar
desteklenmiyor.
                    postfix += " ";
                    i--;
                    continue;
                }
                else if (infix[i] == '(')
                {
                    stack->push(infix[i]);
                }
                else if ((infix[i] == '*') || (infix[i] == '+') || (infix[i] == '-') || (infix[i] == '/'))
                {
                    while ((!stack->isEmpty()) && (stack->top() != '('))
                    {
                        if (OncelikDusukmu(stack->top(), infix[i]))
                        {
                            postfix += stack->top();
                            postfix += " ";
                            stack->pop();
                        }
                        else
                        {
                            break;
                        }
                    }
                    stack->push(infix[i]);
                }
                else if (infix[i] == ')')
                {
                    while ((!stack->isEmpty()) && (stack->top() != '('))
                    {
                        postfix += stack->top();
                        postfix += " ";
                        stack->pop();
                    }
                    if (!stack->isEmpty())
                        stack->pop(); // Sol parantez yığıttan pop ediliyor.
                }
            }
            while (!stack->isEmpty())
            {
                postfix += stack->top();
                postfix += " ";
                stack->pop();
            }
            delete stack;
            return postfix;
        }
```

POSTFIX İFADENİN HESAPLANMASI

İfade soldan sağa okunup operatör okunana kadar yığıta okunmuş sayı push yapılır. Operatör okunduğunda yığıttan iki pop yapılıp, operatör bu iki sayı üzerinde uygulanıp sonuç tekrar yığıta push yapılır. Bu şekilde postfix ifadenin tümü okununcaya kadar işlem devam eder.

Postfix: 25 12 – 10 7 + * 4 /	
Postfix: 12 – 10 7 + * 4 /	
Postfix: – 10 7 + * 4 /	
Postfix: 10 7 + * 4 /	
Postfix: 7 + * 4 /	

Postfix: **+ * 4 /**	Yığıt (Stack) 7 10 13
Postfix: *** 4 /**	Yığıt (Stack) 17 13
Postfix: **4 /**	Yığıt (Stack) 221
	Yığıt (Stack) 55,25 221 / 4

```
double PostfixHesapla(string postfix)
        {
            double sonuc = -1;
            int uzunluk = postfix.length();
            Stack<double>* yigit = new Stack<double>();
            for (int i = 0; i < uzunluk; i++)
            {
                // boşlukları geç
                if (isspace(postfix[i])) continue;
                if (isdigit(postfix[i]))
                {
                    string sayi = "";
                    while (isdigit(postfix[i])) { sayi += postfix[i]; i++; } // Ondalik sayılar
desteklenmiyor.
                    double s;
                    s = atof(sayi.c_str());
                    yigit->push(s);
                    i--;
                    continue;
                }
                else // Operatör ise
                {
                    double sayi2 = yigit->top();
                    yigit->pop();
                    double sayi1 = yigit->top();
                    yigit->pop();
                    switch (postfix[i])
                    {
                        case '+':
                            yigit->push(sayi1 + sayi2);
                            break;
                        case '-':
                            yigit->push(sayi1 - sayi2);
                            break;
                        case '*':
                            yigit->push(sayi1 * sayi2);
                            break;
                        case '/':
                            if (sayi2 == 0) return -1;  // Sıfıra bölünme hatası
                            yigit->push(sayi1 / sayi2);
                            break;
                        default: // Hata Desteklenmeyen operatör
                            return -1;
                            break;
                    }
                }
            }
            if (!yigit->isEmpty()) sonuc = yigit->top();
            yigit->pop();
            delete yigit;
            return sonuc;
        }
```

Hazırlayan
Yrd. Doç. Dr. M. Fatih ADAK