# Stacks


## Indah Agustien Siradjuddin

Struktur Data


---

## Struktur Data

Struktur data merupakan cara programmer untuk merepresentasikan suatu data, dan memudahkan programmer untuk menyelesaikan permasalahan dengan menggunakan struktur data tersebut.

Struktur data **Linear Structure** merupakan struktur data sederhana dimana ketika data ditambahkan maka data tersebut berada pada posisi relatif terhadap data yang lain dari struktur data tersebut. Linear structure ini memiliki dua buah ujung. Dua buah ujung ini memiliki beberapa istilah seperti **left-right**, **front-rear**, ataupun **top-bottom**. Terdapat beberapa struktur data yang termasuk pada linear structure ini, yaitu :

- `Stacks`
- `Queues`
- `Lists`

Ketiga jenis struktur data tersebut berbeda dalam penambahan atau penghapusan suatu data. Misalkan, suatu struktur data hanya diperbolehkan menghapus data pada salah satu ujung, sedangkan pada struktur data lain, penghapusan data dapat dilakukan pada kedua ujung.

Struktur data yang akan dibahas pertama kali adalah *Stacks*, yang terdiri dari :



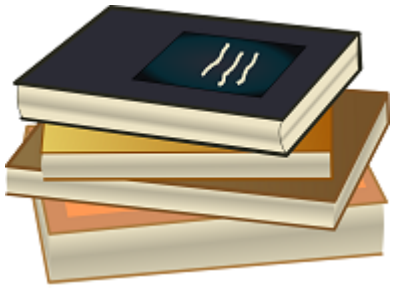
## Definisi Stacks

**Stacks** adalah satu struktur data dimana penambahan dan penghapusan data, hanya dapat dilakukan pada satu ujung yang **sama**, atau yang biasa dikenal dengan istilah **top**.

Semakin data jauh berada dari posisi top, maka data tersebut diindikasikan berada di stack lebih lama dibandingkan dengan data yang berada dekat pada data di posisi top.

Jika terdapat data baru yang ditambahkan di stack, maka data ini pulalah yang akan dihapus ketika terdapat proses penghapusan data. Konsep ini dikenal dengan nama **LIFO-Last In First Out**.

Konsep stack ini dapat ditemui pada permasalahan sehari-hari, misalkan tumpukan buku pada Gambar 1 berikut.



**Gambar 1. Tumpukan buku**

Jika kita ingin mengambil buku pada tumpukan buku tersebut, maka buku yang dapat kita ambil adalah buku yang berada di posisi teratas. Jika ingin mengambil buku yang berada di posisi paling bawah, maka kita harus mengambil buku-buku yang berada di posisi atasnya terlebih dahulu. Begitu juga ketika terdapat penambahan buku baru, maka buku baru ini akan berada di posisi paling atas.

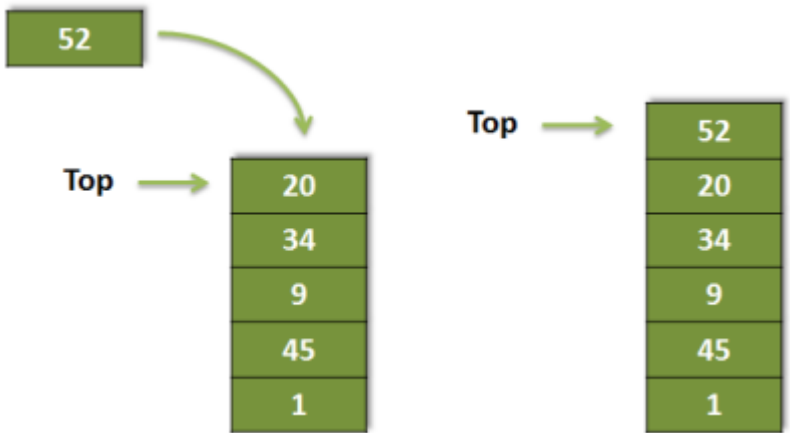
[Kembali ke Menu Awal]

# Operasi pada Stacks

Terdapat operasi dasar pada Stacks,

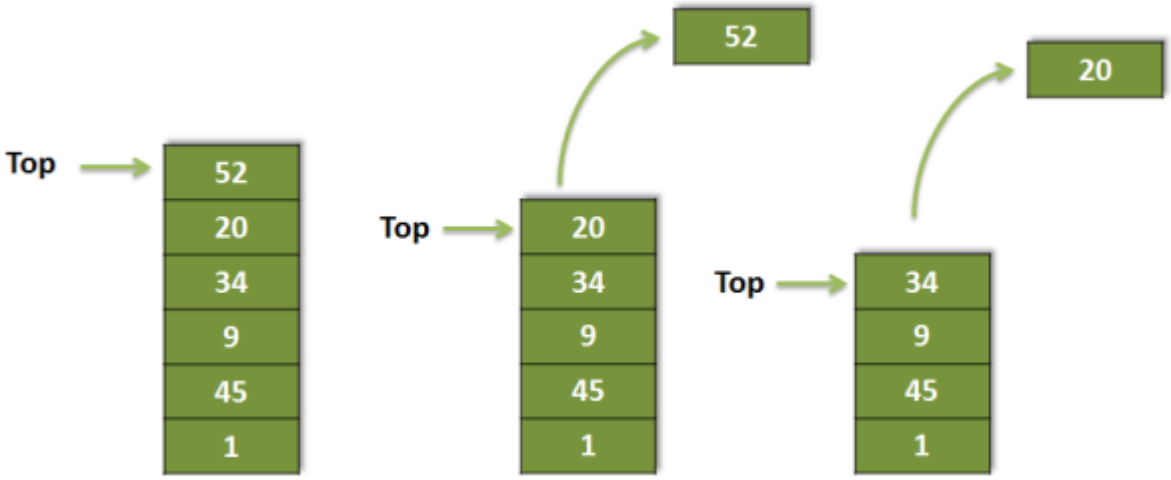
```
- stack (),   inisialisasi stack yang kosong
- push(data), penambahan data baru pada posisi top dari stack
- pop(),      penghapusan data yang terdapat di posisi top dari stack. R
eturn value dari fungsi ini adalah data yang
              dihapus dari stack tersebut
- peek(),     informasi data yang terletak pada posisi top
- isEmpty(), untuk memeriksa apakah stack dalam keadaan kosong
- size(),     informasi jumlah data yang terdapat pada stack
```

Ilustrasi proses penambahan data atau yang dikenal dengan istilah **push** dapat dilihat pada Gambar 2 berikut.



**Gambar 2. Operasi push pada Stacks**

Sedangkan proses penghapusan data atau yang dikenal dengan istilah **pop** dapat dilihat pada Gambar 3 berikut.




**Gambar 3. Operasi pop pada Stacks**

# Code

Berikut adalah fungsi-fungsi yang dibutuhkan untuk mengimplementasikan stacks.

In [1]:
```
def stack():
    s=[]
    return (s)
def push(s,data):
    s.append(data)
def pop(s):
    data=s.pop()
    return(data)
def peek(s):
    return(s[len(s)-1])
def isEmpty(s):
    return (s==[])
def size(s):
    return(len(s))
```

Struktur data direpresentasikan dengan lists. Posisi top dari suatu stacks berada pada posisi terakhir pada list, oleh karena itu, operasi *push* pada stack, menggunakan method *append* dari list, sehingga dengan operasi *append* ini, penambahan data baru terletak pada posisi akhir pada list. . Misalkan terdapat variabel `dataStack` yang menggunakan stacks berisikan data `dataStack=[4,8,1,0]` , maka posisi top berada pada data `0` . Sehingga ketika dilakukan `push(dataStack,72)` , maka `dataStack` akan berubah menjadi `dataStack=[4,8,1,72]` . Sedangkan operasi *pop* pada list, menggunakan method *pop* dari list, karena method *pop* ini menghapus data yang terletak pada posisi akhir dari list.

Berikut adalah contoh penggunaan fungsi-fungsi stacks yang telah dibuat

In [2]:
```
import Stack as st
```

In [3]:
```
data=st.stack()
st.isEmpty(data)
```

Out[3]: True

In [4]:
```
st.push(data,100)
st.push(data,23)
st.push(data,34)
st.pop(data)
st.push(data,56)
st.pop(data)
print(data)
```

```
[100, 23]
```

### Latihan - 1

Buat function untuk 'reverse word' dengan menggunakan konsep **stacks**, misalkan kata 'stacks' menjadi 'skcats'

In [5]:
```
def reverseWord(word):
    dataStack=st.stack()
    for ch in word:
        st.push(dataStack,ch)
    temp=''
    while not(st.isEmpty(dataStack)):
        temp=temp+st.pop(dataStack)
    return temp
```

In [6]:
```
reverseWord('aku')
```

Out[6]: 'uka'

[Kembali ke Menu Awal]

## Implementasi Stacks

Berikut adalah contoh implementasi untuk penggunaan stacks, yaitu *delimiter matching* dan konversi desimal ke biner

### Delimiter Matching

**Parentheses** sering digunakan untuk urutan penyelesaian dalam persamaan Matematika, seperti contoh berikut:

P = 5 x (4 + 5) / ((3 + 2) x (10 - 8))

Dengan penggunaan **parentheses** maka persamaan tersebut dapat diselesaikan secara benar, hanya saja terkadang **parentheses** yang ditulis tidak lengkap, sehingga hasil persamaan tidak dapat dikerjakan atau tidak sesuai dengan yang diinginkan, seperti contoh berikut :

P = 5 x (4 + 5) / ((3 + 2) x (10 - 8)

Oleh karena itu diperlukan algoritma untuk pemeriksaan parentheses ini atau secara umum **delimiter**, yaitu '{ }', '[ ]', dan '( )'.

Beberapa tahapan atau algoritma untuk pengecekan *parentheses* ini adalah :

1. Baca setiap karakter yang terdapat pada suatu string matematika
2. Jika karakter adalah kurung buka, maka masukkan dalam stack
3. Jika karakter adalah kurung tutup, ada beberapa kondisi yang harus diperiksa, yaitu :
   - Jika stack dalam keadaan empty, maka jumlah kurung tutup lebih banyak daripada kurung buku
   - Jika stack dalam keadaan tidak empty, maka maka pop stack, dan cocokkan karakter hasil pop dengan kurung tutup, jika sejenis, maka *matched*, jika tidak maka jenis kurung tidak sama
4. Jika semua karakter telah terbaca, dan stack masih dalam keadaan terisi (tidak *empty*), maka jumlah kurung buka lebih banyak daripada kurung tutup

In [7]:
```
def checkParentheses(strMat):
    data=st.stack()
    for ch in strMat:
        if ch=='(':
            st.push(data,ch)
        elif ch==')':
            if st.isEmpty(data):
                return 'kelebihan kurung tutup'
            else:
                st.pop(data)
    if st.isEmpty(data):
        return True
    else:
        return 'Kelebihan kurung buka'
```

In [8]:
```
checkParentheses('5 x (4 + 5) / (3 + 2) x (10 - 8))')
```

Out[8]: 'kelebihan kurung tutup'

In [9]:
```
a='5 x (4 + 5) / (3 + 2) x (10 - 8))'

'300' in a
```

Out[9]: False

## Code

Berikut adalah fungsi untuk pengecekan kurung pada suatu string matematika

In [10]:
```
def stack():
    s=[]
    return (s)
def push(s,data):
    s.append(data)
def pop(s):
    data=s.pop()
    return(data)
def peek(s):
    return(s[len(s)-1])
def isEmpty(s):
    return (s==[])
def size(s):
    return(len(s))
```

In [11]:
```
def parenthesesCheck(strMath):
    kurung={')':'(','}':'{',']':'['}
    kurungBuka=kurung.values()
    kurungTutup=kurung.keys()
    temp=stack()
    matched=True
    for ch in strMath:
        if ch in kurungBuka: # if ch in kurung.values()
            push(temp,ch)
        if ch in kurungTutup: #if ch in kurung.keys()
            if isEmpty(temp):
                return 'Kelebihan kurung Tutup'
            else:
                tempKurung=pop(temp)
                if tempKurung==kurung[ch]:
                    matched=matched and True
                else:
                    matched=matched and False
    if not(isEmpty(temp)):
        return 'Kelebihan Kurung Buka'
    if matched==True:
        return 'OK'
    else:
        return 'Tidak Cocok kurungnya'
```

In [12]:
```
def convertBinary(num):
    temp=stack()
    divNum=num
    while divNum>0:
        push(temp,divNum%2)
        divNum=divNum//2
    tempStr=''
    for i in range(size(temp)):
        tempStr=tempStr+str(pop(temp))
    return tempStr

print(convertBinary(25))
```

```
11001
```

In [13]:
```
parenthesesCheck('{5+2}*3)')
```

Out[13]:
```
'Kelebihan kurung Tutup'
```

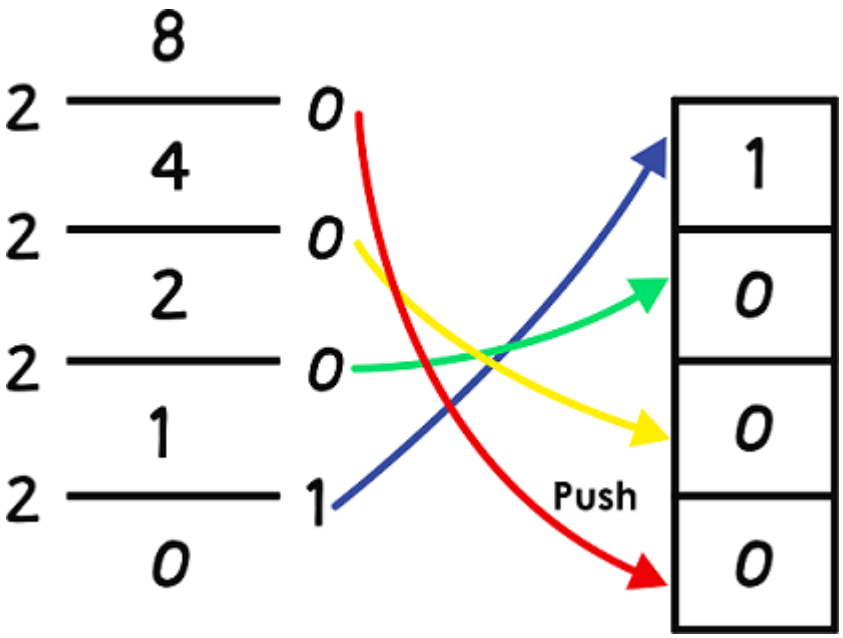
In [14]:
```
def paranthesesCheck(strMath):
    operandStack=stack()
    lenMath=len(strMath)
    openOperand='({['
    closeOperand=')}]'
    #print(lenMath)
    i=0
    Matched=True;
    while i<(lenMath):
        #print(i,'=',strMath[i])
        if strMath[i] in openOperand:
            push(operandStack,strMath[i])
            #print(operandStack)
        elif strMath[i] in closeOperand:
            if not (isEmpty(operandStack)):
                top=pop(operandStack)
                #print("top=",top)
                #print (operandStack)
                if openOperand.index(top)==closeOperand.index(strMath[i]):
                    Matched=Matched and True
                else:
                    Matched=Matched and False
                    print ('Kurung Buka dan Kurang Tutup tidak Cocok')
            else:
                Matched=Matched and False
                print('Jumlah Kurung Tutup lebih banyak')
        i=i+1
        #print(Matched)
    if not(isEmpty(operandStack)):
        Matched=False
        print('Jumlah Kurung Buka Lebih banyak')
    return(Matched)
```

In [15]:
```
a='5 x (4 + 5) / ((3 + 2) x (10 - 8)'
```

```
In [16]: isMatched=paranthesesCheck(a)
         print(isMatched)

         Jumlah Kurung Buka Lebih banyak
         False

In [17]: paranthesesCheck('5 x (4 + 5) / ((3 + 2) x (10 - 8))')

Out[17]: True

In [18]: paranthesesCheck('5 x (4 + 5} / ((3 + 2) x (10 - 8))')

         Kurung Buka dan Kurang Tutup tidak Cocok

Out[18]: False
```

## Konversi Bilangan

Pada dunia ilmu komputer seringkali dibutuhkan proses konversi dari suatu bilangan ke bilangan lain, misalkan dari bilangan desimal menjadi bilangan biner atau menjadi bilangan octal. Ilustrasi proses konversi dari bilangan desimal ke bilangan biner dapat dilihat pada Gambar 4 berikut.




**Gambar 4. Konversi bilangan desimal ke biner**

Dari Gambar tersebut dapat dilihat bahwa bilangan biner didapatkan dari **sisa** pembagian bilangan desimal dengan **dua**. Hasil konversi didapatkan dengan cara membaca terbalik semua sisa tersebut.

## Latihan - 2

Buat function untuk mengkonversi bilangan desimal menjadi bilangan biner dengan menggunakan konsep **stack**

[Kembali ke Menu Awal]

# Ekspressi Aritmatik Infix, Prefix, Postfix

Eksperssi aritmatik yang sering ditemui adalah operasi aritmatik **infix**. Pada operasi aritmatik **infix** ini, **operator** (+,-,*,/, dll) berada diantara dua buah **operand** (bilangan). Ekspressi aritmatik ini dapat dengan mudah diselesaikan ketika tidak ada ambiguitas, seperti:

$$A + B$$
$$A * X$$
$$A - B + C$$

Akan tetapi, ekspressi aritmatik ini akan sulit diselesaikan jika terjadi ambiguitas, seperti ekspressi aritmatik berikut ini:

$$A + B * C$$

Pada ekspressi tersebut, terdapat ambiguitas, yaitu apakah A + B diselesaikan terlebih dahulu ataukah B * C yang harus diselesaikan terlebih dahulu. Ambiguitas ini dapat diatasi dengan konsep **operator precedence**. Pada operator precedence, operator-operator dibagi menjadi beberapa level berdasarkan prioritas urutan penyelesaian.
Pada contoh ekspressi aritmatik A + B * C, berdasarkan operator precedence, maka urutan penyelesaian adalah B * C kemudian hasilnya ditambahkan dengan A. Permasalahan terjadi ketika pada ekspressi aritmatik A + B * C, A + B ingin diselesaikan terlebih dahulu. Untuk penyelesaian ekspressi aritmatik yang tidak sesuai dengan urutan operator precedence, maka tanda '( )' sangatlah diperlukan, sehingga operasi tersebut menjadi (A + B) * C.

Tanda '( )' **tidaklah diperlukan** pada ekspressi aritmatik **prefix** dan **postfix**. Pada ekspressi aritmatik **prefix**, operator mendahului operand, sedangkankan ekspressi aritmatik **postfix**, operator ditulis setelah operand.

Berikut contoh ekspressi aritmatik **infix** dan konversinya ke **prefix** dan **postfix**.

| Infix | Prefix | Postfix |
|---:|---:|---:|
| A+B | +AB | AB+ |
| (A+B)*C | *+ABC | AB+C* |
| (A+B)*(C-D) | *+AB-CD | AB+CD-* |
| A+B*C-D | -+A*BCD | ABC*+D- |
| A+B-C+D | +-+ABCD | AB+C-D+ |
| A*B-C*D | -*AB*CD | AB*CD*- |

## Konversi Infix ke Postfix

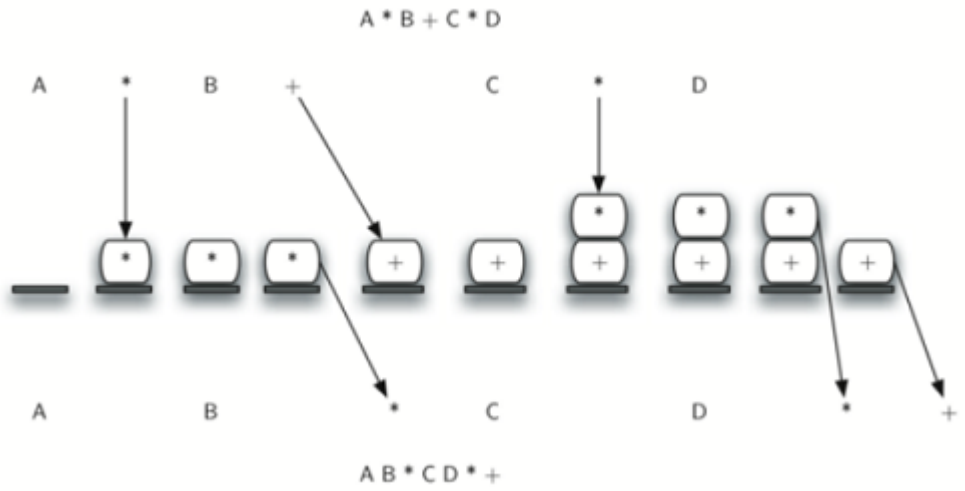
Berikut algoritma untuk konversi ekspressi aritmatik *infix* ke *postfix* :

1. Buat struktur data *stack* untuk menampung operator
2. Baca ekspressi aritmatik dari kiri ke kanan tiap token :

- jika token yang dibaca adalah *operand* maka masukkan *operand* tersebut ke dalam *output string*
- jika token yang dibaca adalah kurung buka maka *push* kurung buka tersebut ke dalam *stack*
- jika token yang dibaca adalah kurang tutup maka *pop* stack semua token sampai ditemukan kurung buka
- jika token yang dibaca adalah *operator* maka :
  - pop operator-operator yang memiliki precedence lebih tinggi atau sama dan masukkan operator tersebut ke dalam *output string*
  - push token operator ke dalam stack

3. Jika masih terdapat operator pada stack, maka pop operator yang tersisa dan letakkan pada *output string*

Contoh konversi dari infix ke postfix dapat dilihat pada Gambar 5.




**Gambar 5. Konversi bilangan desimal ke biner**

## Latihan - 3

Buat function untuk mengkonversikan ekspressi aritmatik infix menjadi ekspressi aritmatik postfix

## Evaluasi Ekspressi Postfix

Seperti yang sudah dijelaskan sebelumnya, evaluasi untuk ekspressi aritmatika Postfix dilakukan setelah terdapat dua buah operand sebelum sebuah operator. Misalkan : 87+2*, yang berarti (8+7)*2=30, oleh karena itu berikut algoritma untuk mengevaluasi ekspressi aritmatika postfix :

1. baca string mulai dari kiri
2. Jika character yang dibaca adalah sebuah operand, maka push operand tersebut.
3. Jika character yang dibaca adalah sebuah operator, maka pop dua buah operand yang terdapat pada stack, operasikan dua buah operand tersebut dengan operator, dan push hasil operasinya
4. hasil akhir adalah angka terakhir yang terdapat di dalam stack

## Code

Berikut code untuk evaluasi ekspressi postfix

In [19]:
```
def evaluatePost(postStr):
    operandStack=stack()
    operator='+-/*'
    for i in postStr:
        if i not in operator:
            push(operandStack,i)
        else:
            oprnd2=pop(operandStack)
            oprnd1=pop(operandStack)
            if i=='+':
                result=float(oprnd1)+float(oprnd2)
            elif i=='-':
                result=float(oprnd1)-float(oprnd2)
            elif i=='*':
                result=float(oprnd1)*float(oprnd2)
            else:
                result=float(oprnd1)/float(oprnd2)
            push(operandStack,result)
    return(pop(operandStack))
```

In [20]:
```
print(evaluatePost('45-6*'))
```

```
-6.0
```

[Kembali ke Menu Awal]