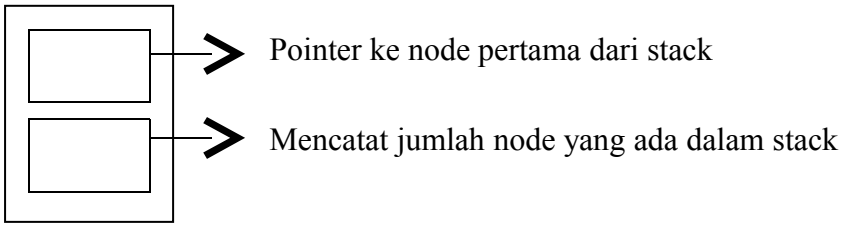
Stack

Sekilas Tentang Stack

Stack dapat diimplementasikan dengan menggunakan konsep *Linked List*. Bentuk dari *struct* dari head untuk stack dapat digambarkan sebagai berikut:



Perbedaan mendasar antara *stack* dan *Linked List* adalah bahwa *stack* memiliki operasi khusus untuk memanipulasi elemen dalam stack tersebut. Operasi yang dimaksud dalam program C dapat diimplementasikan menjadi function. Operasi yang dimaksud adalah:

- push() : menambah elemen stack yang baru dibagian atas stack.
- pop() : menghapus elemen dari stack yang berada dibagian paling atas.
- get() : membaca nilai elemen dari stack yang berada dibagian paling atas.
- isEmpty() : memeriksa apakah stack kosong atau tidak.

Kegunaan Stack

Stack biasa digunakan dalam mengontrol operasi dalam sebuah sistem operasi. Selain itu stack juga merupakan algoritma yang baik yang dapat digunakan untuk membuat phaser (membaca urutan operasi dari sebuah persamaan matematika).

Implementasi Stack Untuk Integer

```
typedef struct stacknode * StackNodePtr;
typedef struct stacknode
{
      int         elemen;
      StackNodePtr     next;
}StackNode;

typedef struct stack
{
      StackNodePtr     top;
      unsigned   size;
}Stack;
```

Bagian Deklarasi

Bagian deklarasi di atas kita asumsikan disimpan menjadi sebuah *header file* dengan nama *stack.h*

Function-function dibawah ini kita asumsikan disimpan dalam *stack.c*

```
#include "stack.h"
#include <stdio.h>
int main(void) {
      Stack st;

      makeStack(&st);
      push(&st, 2);
      push(&st, 4);
      push(&st, 3);
      while (!emptyStack(&st))  {
            printf("%d\n", get(&st));
            pop(&st);
      }
      freeStack(&st);
      return EXIT_SUCCESS;
}

int  makeStack(Stack  * ps)  {
      ps->top = NULL;
      ps->size = 0;
      return 1;                  /* sukses */
}

int emptyStack(Stack * ps)  {
      return ps->top == NULL
}

int push(Stack * ps, int elemen)  {
      StackNodePtr  new;
      new = malloc(sizeof(StackNode));
      if(new==NULL)
            return 0;            /* alokasi memori gagal */
      new->elemen = elemen;
      new->next = ps->top;
      ps->top = new;
      ps->size++;
      return 1                   /* sukses */
}

int pop(Stack * ps) {
      StackNodePtr  temp;
      if(ps->top == NULL)
            return 0;            /* stack kosong */
      temp = ps->top;
      ps->top = ps->top->next;
      free(temp);
      ps->size--;
      return 1;                  /* sukses */
}

int get(Stack * ps)  {
      return ps->top->elemen;
}
```

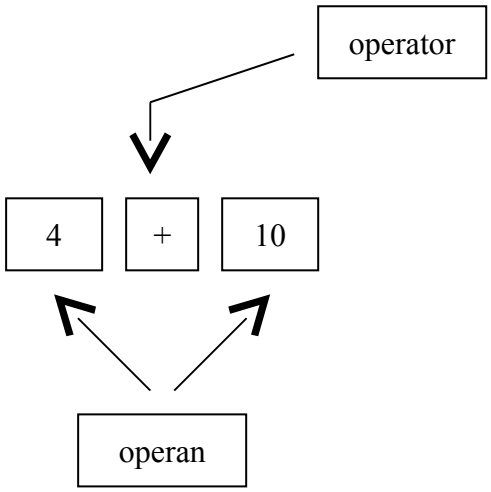
Aplikasi Stack

Salah satu aplikasi stack adalah dalam mengevaluasi dan menghitung ekspresi aritmatik. Sebagai contoh, bila kita ingin mencari nilai dari suatu ekspresi aritmatik sederhana yang melibatkan perkalian, penjumlahan, pengurangan dan pembagian. Eksepresi aritmatik berikut ini dikenal dengan susunan ekspresi *infix*:

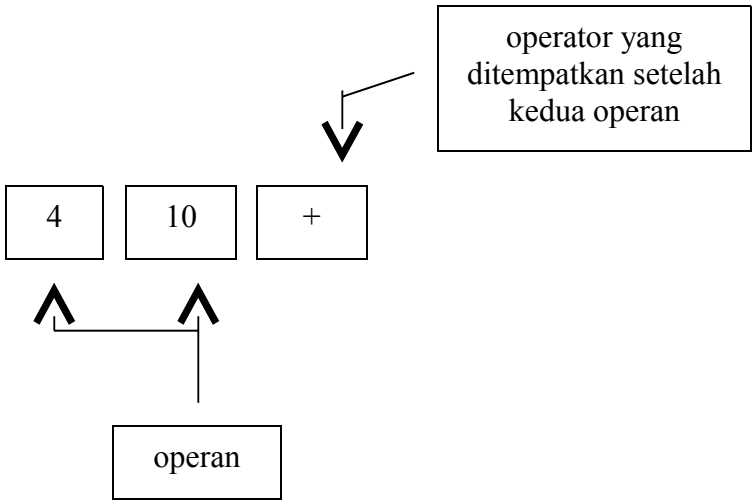
15 * (((3 - 2) * (4 + 10)) / 7)

membutuhkan penyimpanan hasil sebagian perhitungan. Misalnya, jika bagian 3 - 2 pada ekspresi di atas dihitung, maka kita harus menyimpan hasil pengurangan itu yaitu 1 terlebih dahulu, dan kemudian menghitung ekspresi 4 + 1-. Dalam hal ini stack sangat ideal untuk digunakan.

Sebelum kita menyelesaikan permasalahan di atas, mari kita coba menyederhanakan ekspresi *infix* di atas menjadi dalam susunan ekspresi *postfix*. Ekspresi *postfix* adalah *ekspresi yang menempatkan operator setelah kedua operannya*. Berikut ilustrasi dari ekspresi *infix* 4 + 10 yang diubah menjadi ekspresi postfix.



Diubah dalam ekspresi postfix menjadi sebagai berikut:



Oleh karena itu ekspresi aritmatik

15 * (((3 - 2) * (4 + 10)) / 7)

dapat diubah menjadi ekspresi postfix sebagai berikut:

15 3 2 - 4 10 + * 7 / *

Algoritma Mengubah Ekspresi Infix menjadi Postfix

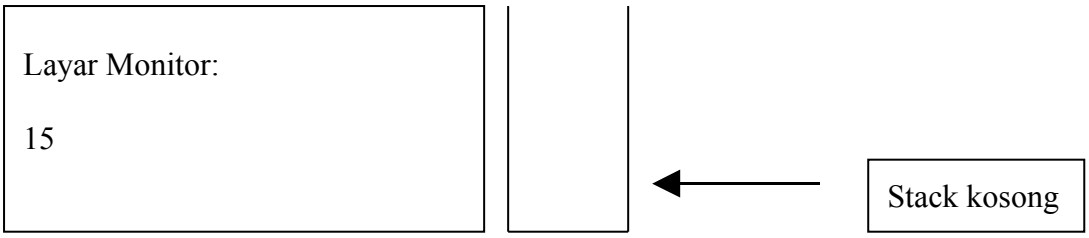
Proses konversi ekspresi infix menjadi bentuk postfix selalu dimulai dari sebelah kiri ekspresi aritmatik.

1. Jika yang ditemukan adalah operan, maka cetak atau simpan
2. Jika yang ditemukan adalah kurung buka maka abaikan.
3. Jika yang ditemukan adalah operator maka *push* ke dalam stack.
4. Jika yang ditemukan adalah kurung tutup maka pop stack, kemudian cetak atau simpan nilainya.
5. Jika ekspresi telah diperiksa semua tetapi ternyata stack belum kosong, pop semua data dalam stack dan cetak atau simpan nilai yang di pop.

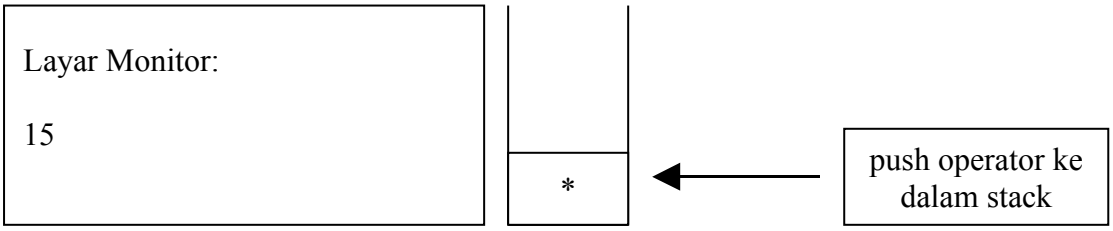
Perhatikan contoh berikut. Diketahui ekspresi infix berikut ini:

15 * (((3 - 2) * (4 + 10)) / 7)

Dimulai dari sebelah kiri, maka yang kita temukan pertama sekali adalah nilai 15 (operan) maka aturan nomor 1 harus dijalankan, yaitu nilai tersebut langsung dicetak. Dalam contoh ini kita tidak menyimpan hasil konversi melainkan langsung mencetaknya ke layar monitor.

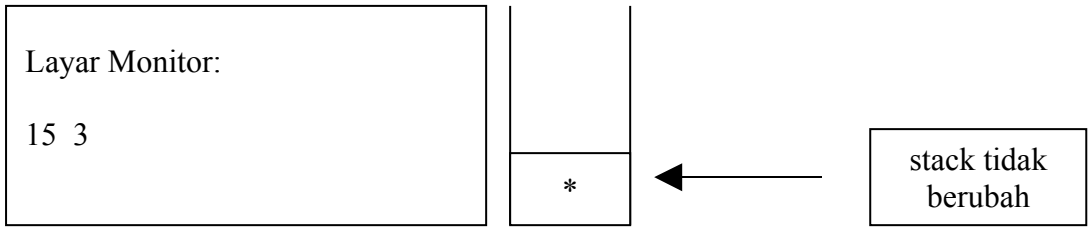


Proses kita lanjutkan, dan kita menemukan operator * maka aturan ke 3 harus dikerjakan, yaitu *push* operator tersebut ke dalam stack.

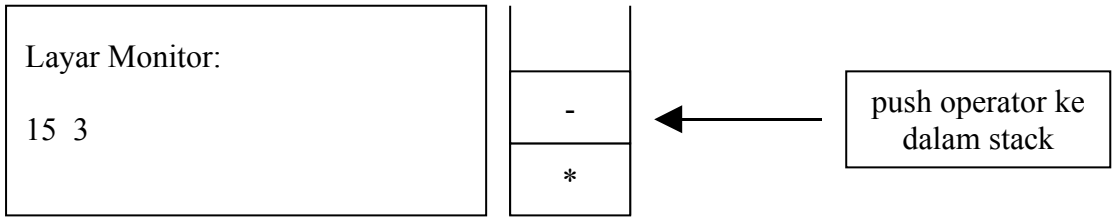


Kemudian kurung buka (ditemukan, maka aturan nomor 2 dikerjakan yaitu abaikan tanda kurung buka tersebut. Pengabaian tanda kurung ini kita lakukan 3 kali karena kita menemukan 3 buah tanda kurung buka.

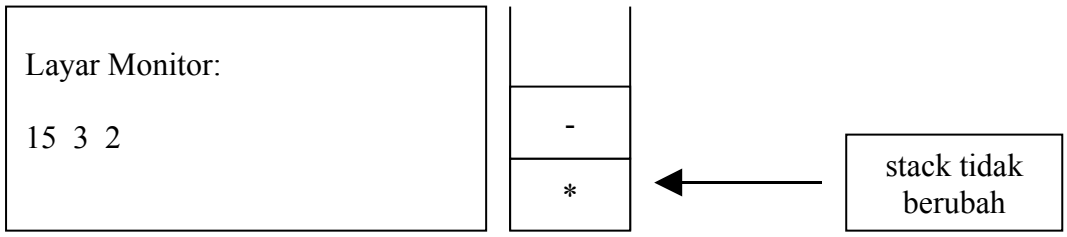
Selanjutnya kita temukan bilangan 3 (operan) maka aturan nomor 1 harus dikerjakan yaitu mencetak bilangan 3 tersebut ke layar.



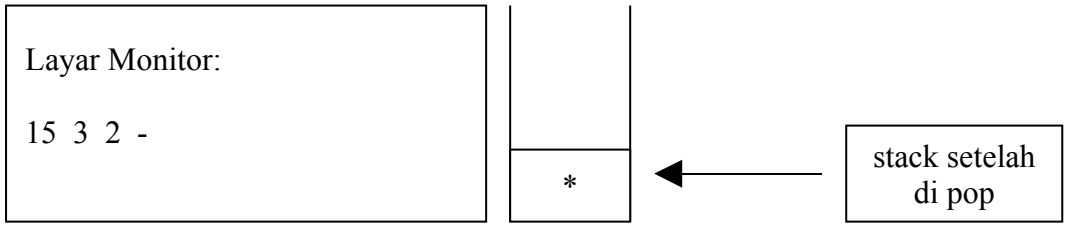
Proses kita lanjutkan, dan kita menemukan operator - maka aturan ke 3 harus dikerjakan, yaitu *push* operator tersebut ke dalam stack.



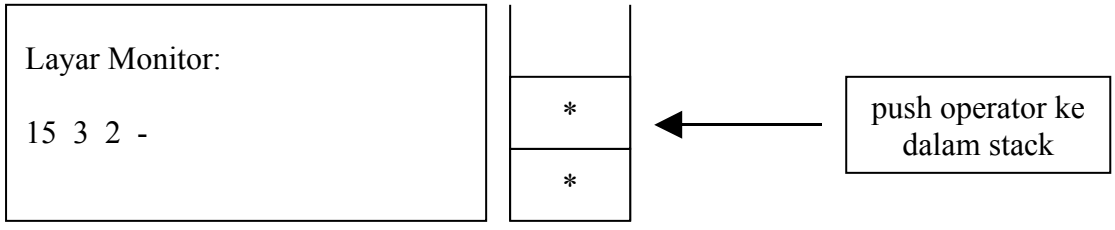
Selanjutnya kita temukan bilangan 2 (operan) maka aturan nomor 1 harus dikerjakan yaitu mencetak bilangan 2 tersebut ke layar.



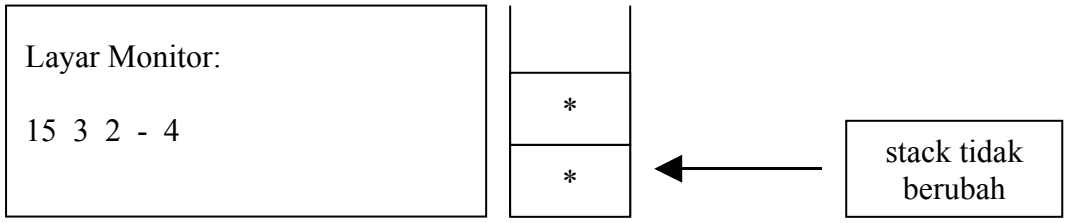
Kemudian tanda kurung tutup) kita temukan sehingga aturan ke 4 harus dikerjakan yaitu pop stack dan cetak data paling atas stack ke layar.



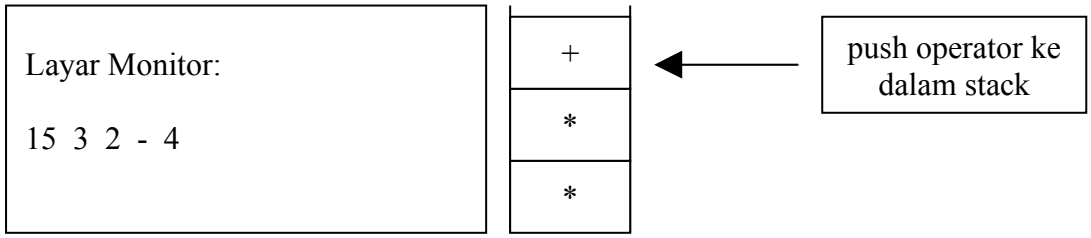
Proses kita lanjutkan, dan kita menemukan operator * maka aturan ke 3 harus dikerjakan, yaitu *push* operator tersebut ke dalam stack.



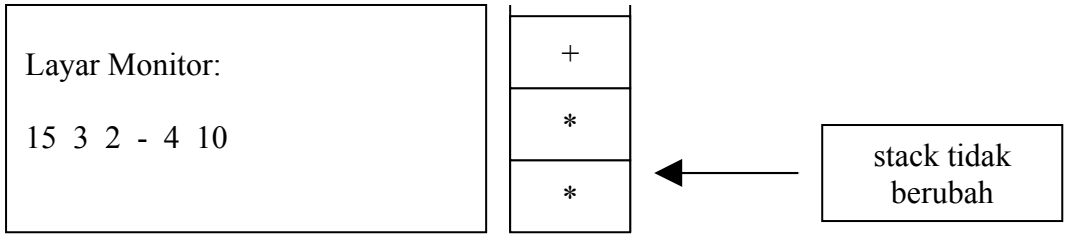
Kemudian kurung buka (ditemukan, sehingga abaikan saja. Selanjutnya kita temukan bilangan 4 (operan) maka aturan nomor 1 harus dikerjakan yaitu mencetak bilangan 4 tersebut ke layar.



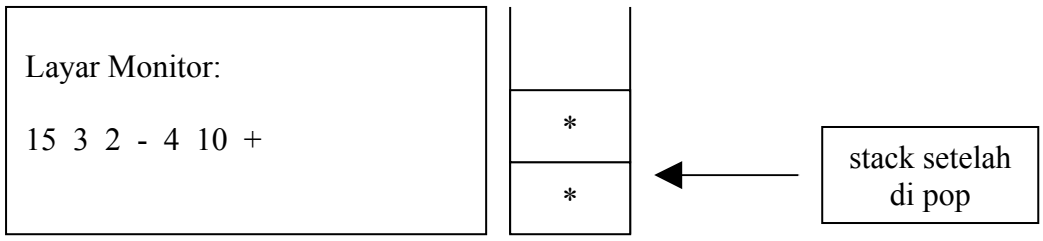
Proses kita lanjutkan, dan kita menemukan operator + maka aturan ke 3 harus dikerjakan, yaitu *push* operator tersebut ke dalam stack.



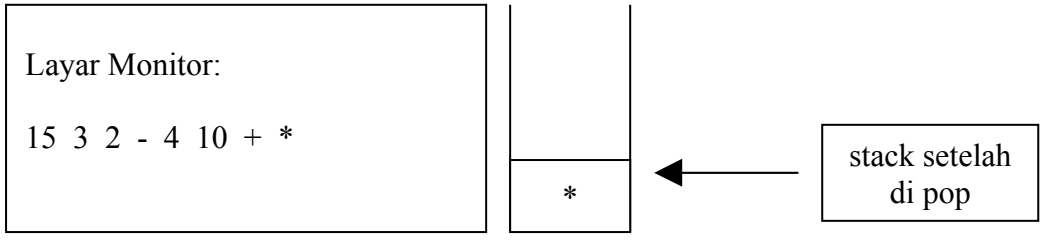
Selanjutnya kita temukan bilangan 10 (operan) maka aturan nomor 1 harus dikerjakan yaitu mencetak bilangan 10 tersebut ke layar.



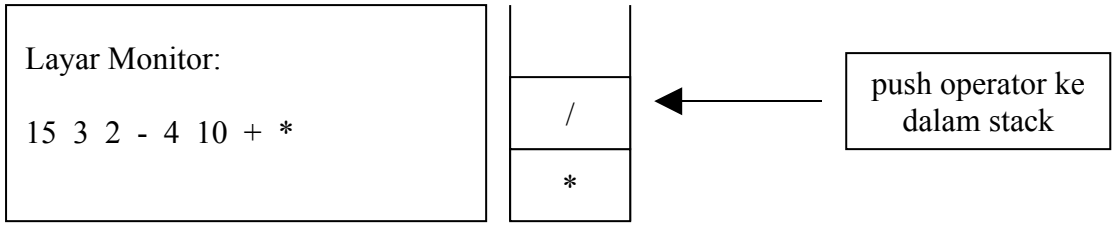
Kemudian tanda kurung tutup) kita temukan sehingga aturan ke 4 harus dikerjakan yaitu pop stack dan cetak data paling atas stack ke layar.



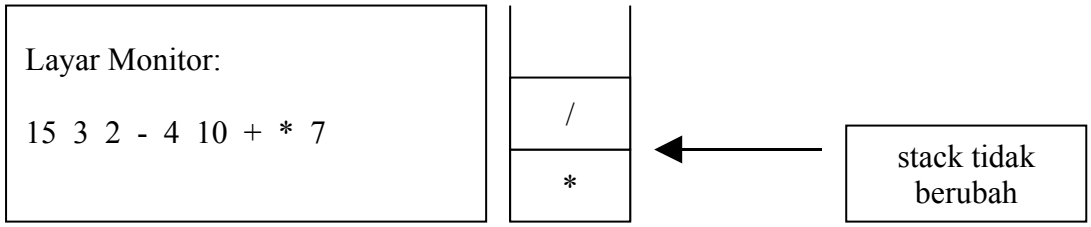
Sekali lagi tanda kurung tutup) kita temukan sehingga aturan ke 4 dikerjakan.



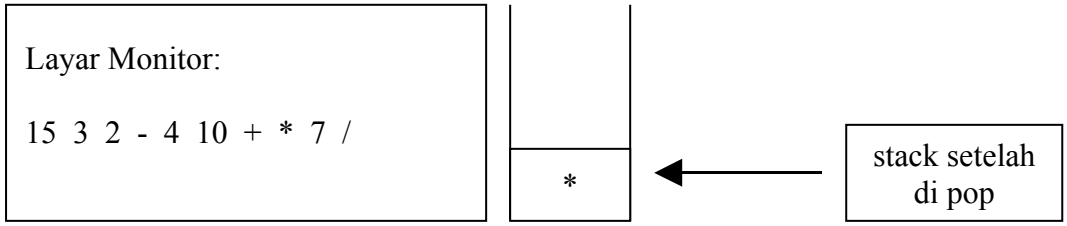
Proses kita lanjutkan, dan kita menemukan operator / maka aturan ke 3 harus dikerjakan, yaitu *push* operator tersebut ke dalam stack.



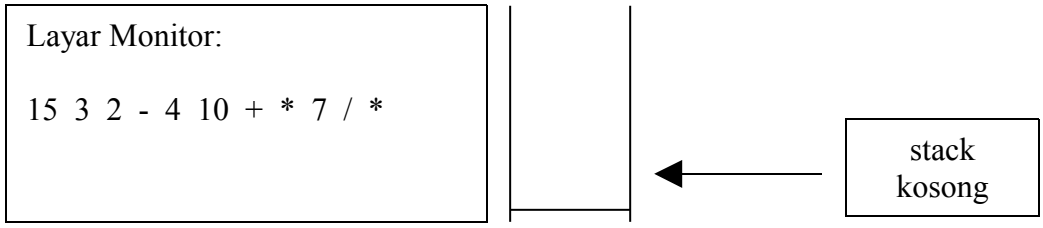
Kemudian kita temukan bilangan 7 (operan) maka aturan nomor 1 harus dikerjakan yaitu mencetak bilangan 7 tersebut ke layar.



Kemudian tanda kurung tutup) kita temukan sehingga aturan ke 4 harus dikerjakan yaitu pop stack dan cetak data paling atas stack ke layar.



Karena ekspresi sudah diperiksa semua sementara stack masih belum kosong maka aturan ke 5 harus di kerjakan yaitu pop semua data dalam stack dan cetak ke layar.



Akhirnya ekspresi postfix kita peroleh yaitu:

15 3 2 - 4 10 + * 7 / *

Algoritma Menghitung Nilai Ekspresi Postfix

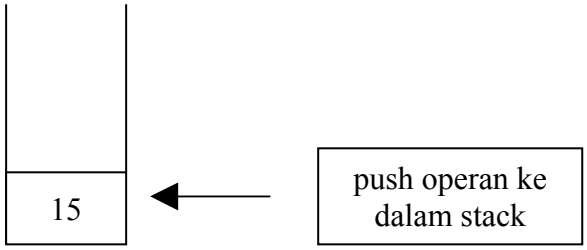
Setelah ekspresi *postfix* kita peroleh, maka beberapa langkah selanjutnya perlu juga dikerjakan untuk menghitung nilai dari ekspresi aritmatik tersebut. Ingat, bahwa sebelum menghitung nilai dari suatu ekspresi aritmatik maka ekspresi *infix* harus diubah terlebih dahulu menjadi bentuk ekspresi *postfix*. Berikut algoritma untuk menghitung nilai ekspresi postfix:

1. Jika yang ditemukan adalah operan, maka push operan tersebut ke dalam stack.
2. Jika yang ditemukan adalah operator, pop stack dua kali dan hitung menggunakan nilai yang di pop dan operator yang dijumpai. Hasil perhitungan ini kemudian di push kembali ke dalam stack.
3. Jika ekspresi telah diperiksa semua maka nilai yang di pop terakhir dari stack adalah jawaban dari ekspresi aritmatik tersebut.

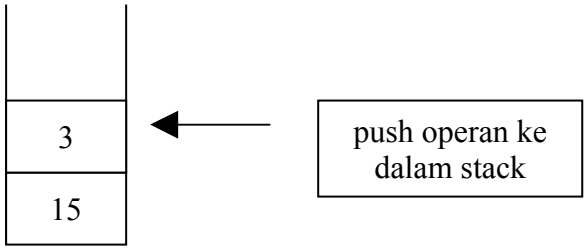
Perhatikan contoh berikut. Jika telah diperoleh ekspresi *postfix* berikut ini:

15 3 2 - 4 10 + * 7 / *

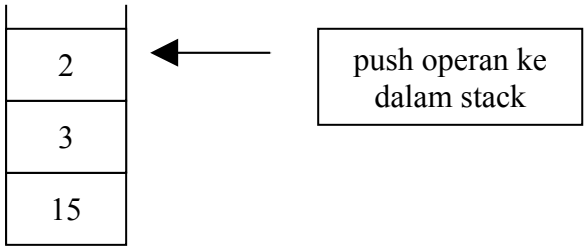
Dimulai dari sebelah kiri, maka yang kita temukan pertama sekali adalah nilai 15 (operan) maka aturan nomor 1 harus dijalankan, yaitu push nilai tersebut ke dalam stack.



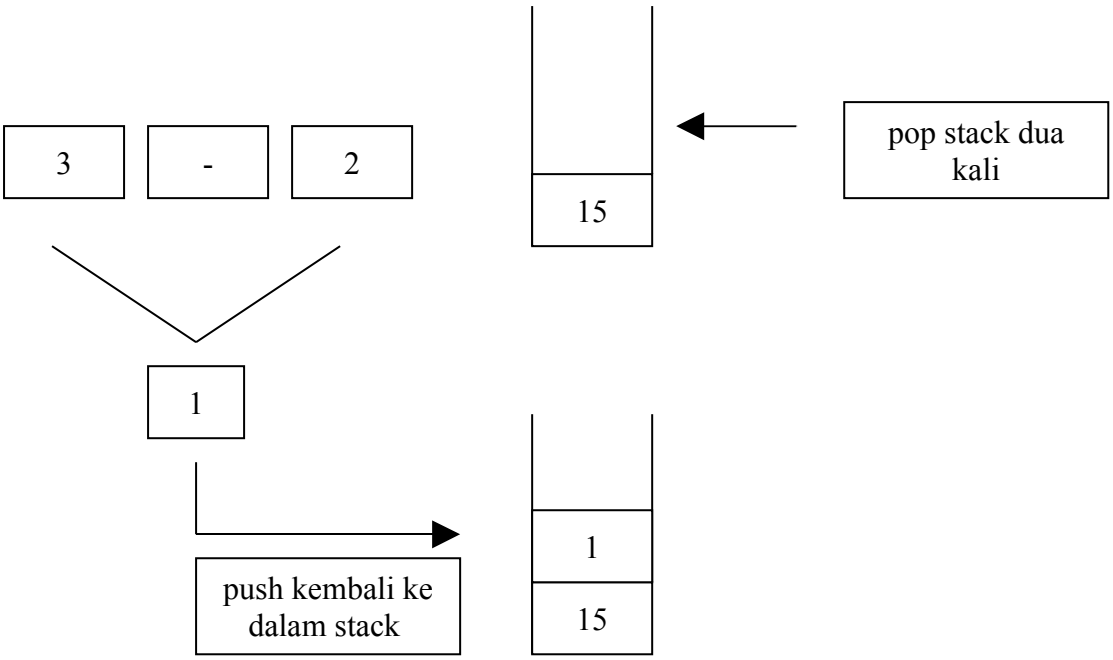
Kemudian kita temukan kembali operan dengan nilai 3, maka aturan nomor 1 harus dikerjakan lagi, yaitu push nilai tersebut ke dalam stack.



Selanjutnya ditemukan kembali operan dengan nilai 2, maka aturan nomor 1 harus dikerjakan lagi, yaitu push nilai tersebut ke dalam stack.

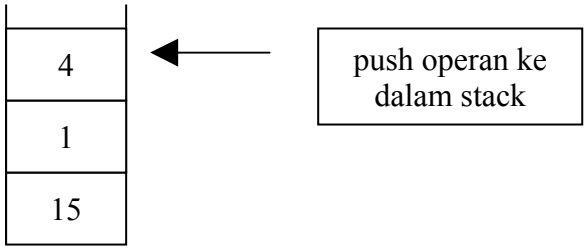


Proses kita lanjutkan lagi, dan sekarang kita temukan operator – oleh karena itu aturan nomor 2 harus dikerjakan.

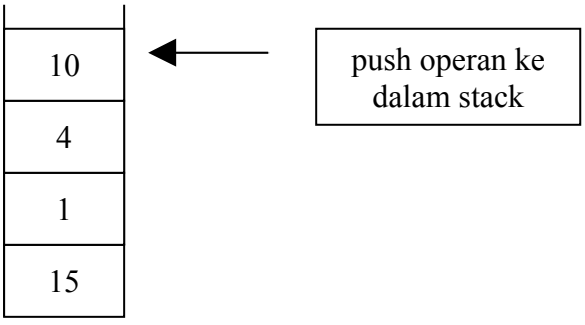


Note: Urutan perhitungan harus 3 – 2 bukan 2 – 3 atau data yang kedua yang di pop di kurang data yang pertama sekali di pop dari stack.

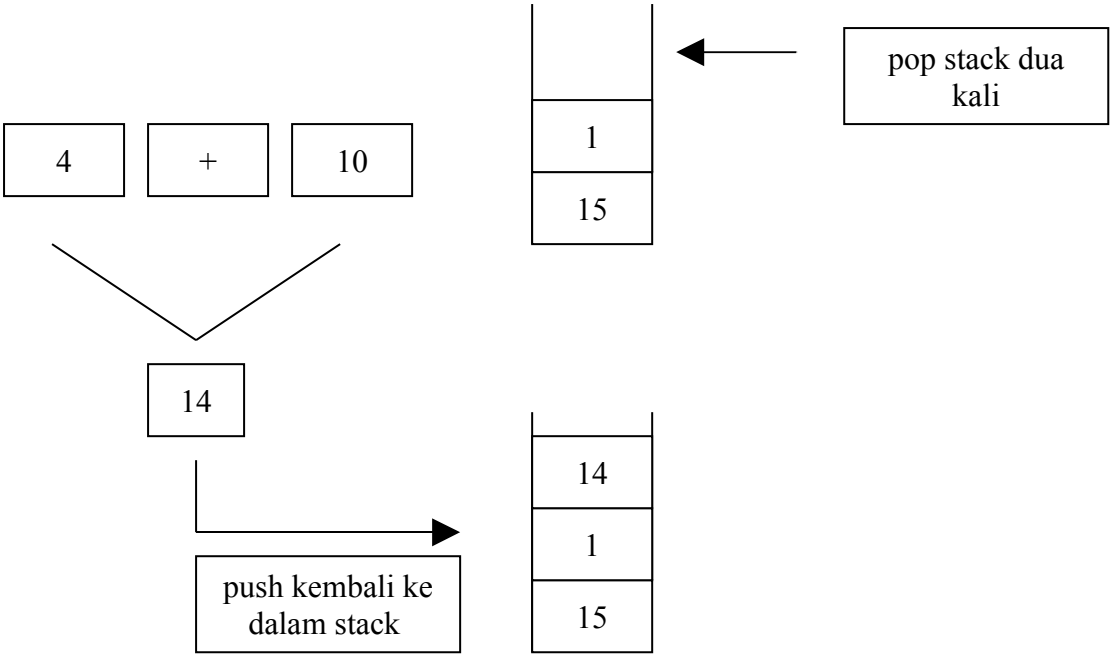
Kemudian ditemukan kembali operan dengan nilai 4, maka aturan nomor 1 harus dikerjakan lagi, yaitu push nilai tersebut ke dalam stack.



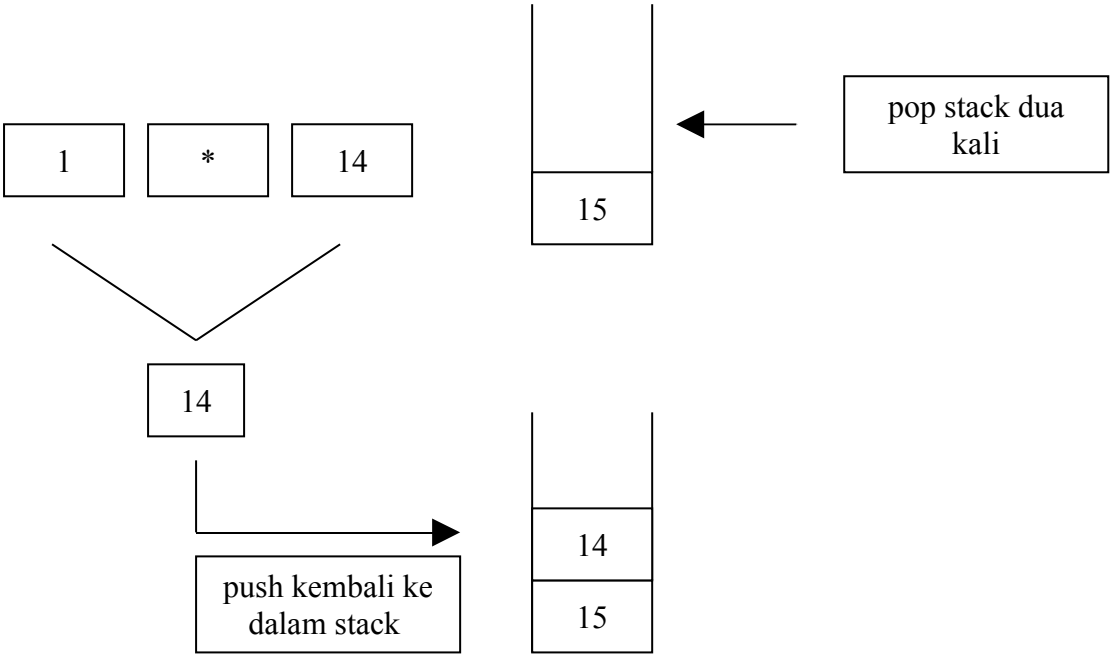
Selanjutnya ditemukan operan dengan nilai 10, maka kembali aturan nomor 1 dikerjakan lagi, yaitu push nilai tersebut ke dalam stack.



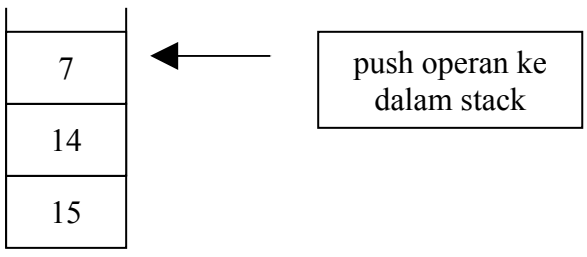
Proses kita lanjutkan lagi, dan sekarang kita temukan operator + oleh karena itu aturan nomor 2 harus dikerjakan.



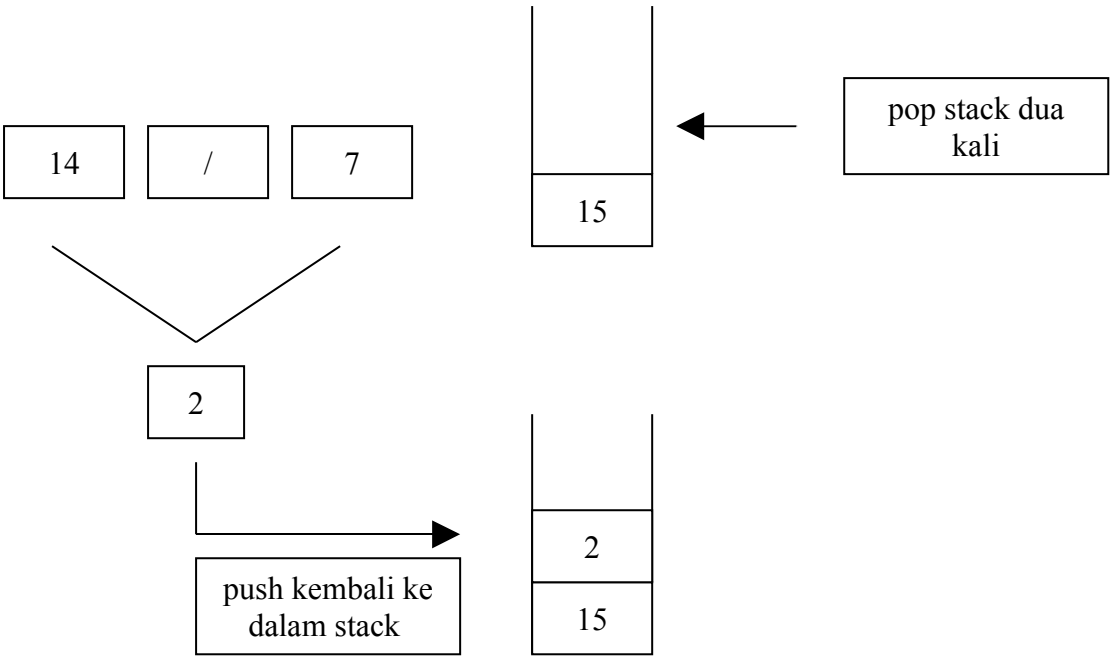
Kemudian kita temukan lagi operator * oleh karena itu aturan nomor 2 dikerjakan.



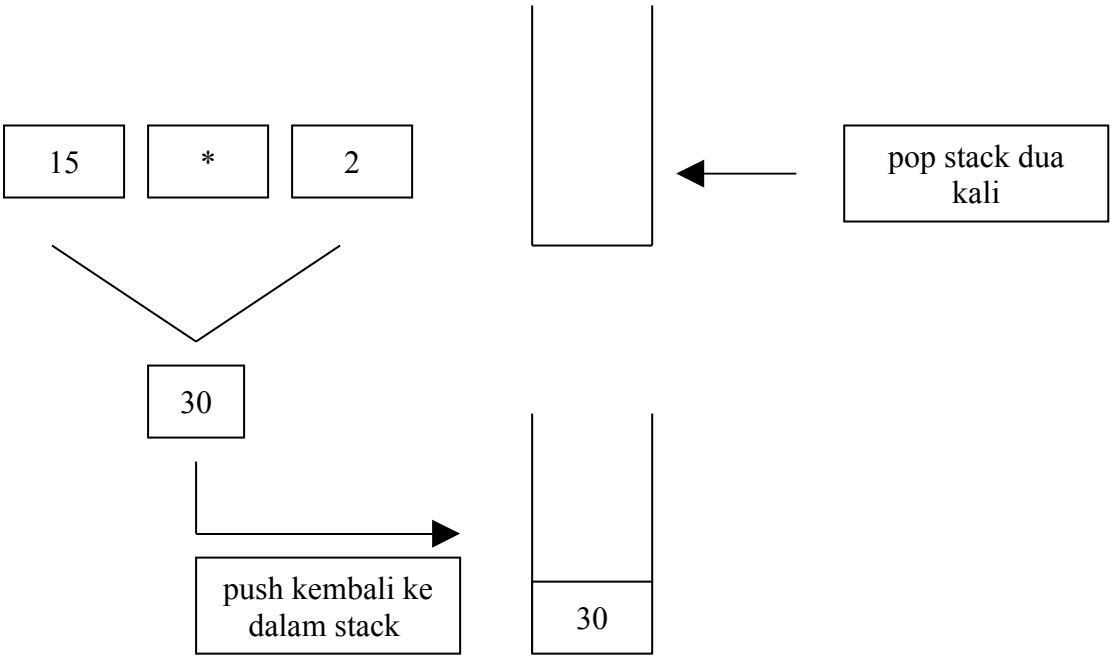
Selanjutnya ditemukan kembali operan dengan nilai 7, maka aturan nomor 1 harus dikerjakan lagi, yaitu push nilai tersebut ke dalam stack.



Proses kita lanjutkan lagi, dan sekarang kita temukan operator / oleh karena itu aturan nomor 2 harus dikerjakan.



Kemudian kita temukan operator * oleh karena itu aturan nomor 2 dikerjakan lagi.



Karena ekspresi *postfix* telah diperiksa semua dan bila data dalam stack di pop maka nilai 30 diperoleh, maka dapat disimpulkan bahwa ekspresi *infix*:

$$15 * (((3 - 2) * (4 + 10)) / 7) = 30$$