Laporan Program Kalkulator

*Dokumen Spesifikasi Program untuk Tugas Akhir Semester MK Struktur Data adn Algoritma (Praktik)*



Disusun Oleh :

|  |  |
| --- | --- |
| Alifah Fisalsabilawati | 201511035 |
| Faishal Muhammad | 201511039 |

Kelas 1 – B

**Jurusan Teknik Komputer dan Informatika Program Studi D3 Teknik Informatika**

**Politeknik Negeri Bandung 2021**

**TABEL REVISI**

|  |  |  |  |
| --- | --- | --- | --- |
| **No** | **Tanggal** | **Keterangan** | **PIC** |
| 1 | 15 Juni 2021 | 1. Menambahkan deskripsi dan fungsi kalkulator pada sub bab deskripsi program 2. Memahami cara mengubah infix ke postfix menggunakan stack 3. Memahami cara pengisian operasi pada stack |  |
| 2 | 18 Juni 2021 | 1. Membuat representasi data 2. Menambahkan proses perubahan infixs ke postfiks menggunakan binary tree |  |
| 3 | 21 Juni 2021 | 1. Menambahkan kamus data |  |
| 4 | 23 Juni 2021 | 1. Memperbaiki representasi data 2. Memperjelas identifikasi proses 3. Memperbaiki kamus data |  |
| 5 | 29 Juni 2021 | 1. Melengkapi kamus data 2. Menambahkan modul yang akan dipakai pada program 3. Menambahkan algoritma proses 4. Menambahkan structured chart 5. Melengkapi identifikasi proses |  |
| 6. | 07 Juli 2021 | 1. Menambahkan gambaran proses perhitungan menggunakan stack 2. Menambahkan proses penetuan postfix menggunakan traversal postorder 3. Melengkapi kamus data 4. Melengkapi representasi data |  |

**DAFTAR ISI**

[BAB I 4](#_Toc75857353)

[1.1 Deskripsi Program 4](#_Toc75857354)

[1.2. Identifikasi Proses 10](#_Toc75857357)

[1.3. Identifikasi Data 11](#_Toc75857358)

[BAB II 13](#_Toc75857359)

[2.1 Perancangan proses 13](#_Toc75857360)

[2.1.1 Structured Chart 13](#_Toc75857361)

[2.1.2 Penjelasan Modul 13](#_Toc75857362)

[2.1.3 Algoritma Proses 13](#_Toc75857363)

[BAB III 14](#_Toc75857364)

[DAFTAR PUSTAKA 15](#_Toc75857365)

[DAFTAR KONTRIBUSI ANGGOTA KELOMPOK 16](#_Toc75857366)

### BAB I

**SPESIFIKASI PROGRAM**

## Deskripsi Program

1. **Definisi**

Kalkulator adalah alat untuk menghitung dari perhitungan sederhana seperti [penjumlahan](https://id.wikipedia.org/wiki/Penjumlahan), [pengurangan](https://id.wikipedia.org/wiki/Pengurangan), [perkalian](https://id.wikipedia.org/wiki/Perkalian) dan [pembagian](https://id.wikipedia.org/wiki/Pembagian) sampai kepada kalkulator  [sains](https://id.wikipedia.org/wiki/Sains)

yang dapat menghitung [rumus](https://id.wikipedia.org/wiki/Rumus) [matematika](https://id.wikipedia.org/wiki/Matematika) tertentu. Pada perkembangannya sekarang ini, kalkulator sering dimasukkan sebagai [fungsi](https://id.wikipedia.org/wiki/Fungsi) tambahan daripada [komputer](https://id.wikipedia.org/wiki/Komputer), [handphone](https://id.wikipedia.org/wiki/Handphone), bahkan sampai [jam tangan](https://id.wikipedia.org/wiki/Jam_tangan). Rata-rata kalkulator memiliki 12 hingga 15 digit, tetapi mesin hitung ilmiah sanggup menghitung hingga angka 1 [googol](https://id.wikipedia.org/wiki/Googol) menggunakan rumus tertentu.

Dalam program yang akan dibuat merupakan kalkulator basic yang biasa digunakan pada umumnya. Didalamnya kami menggunakan operasi infix, prefix dan postfix dengan struktur list stack dan struktur data tree.

1. **Konsep Program**

Pertama yang dilakukan yaitu mengubah operasi infix menjadi postfix. Dalam merubahnya menggunakan 2 cara, yang pertama menggunakan cara manual dan menggunakan stack. Berikut contoh notasi infix

* Cara stack

*Stack* bisa dibaratkan sebagai tumpukan . Stack memiliki cara kerja, “yang pertama masuk ke kotak, maka akan terakhir kali diambil kembali” atau “*first in* *last out*”*,* atau sebaliknya, “yang terakhir masuk ke kotak, akan diambil yang pertama kali,”

atau “*last in first out*.”

1. Proses yang akan dilakukan dari kiri ke kanan
2. Kondisi stack masih NULL
3. Jika yang diproses pertama adalah operand, maka tulis di hasil. Di contoh operand adalah “5”

5 \* (2 + 3 ) / 1 – 6

NULL

Hasil = 5

Stack

1. Lanjutkan ke operator “\*”, karena kondisi stack masih NULL, maka masukkan operator tersebut ke dalam stack

5 \* ( 2 + 3 ) / 1 – 6

\*

5

1. Lanjutkan ke operator “(”, lalu masukkan ke dalam stack saja

5 \* ( 2 + 3 ) / 1 – 6

5

(

\*

1. Lanjutankan ke operand 2, karena operand maka 2 dijadikan hasil saja

5 \* ( 2 + 3 ) / 1 – 6



(

\*

5 2

1. Lanjutkan ke operator “+”, operator ini dimasukkan ke stack saja

5 \* ( 2 + 3 ) / 1 – 6

+

(

\*

5 2

1. Lanjutankan ke operand 3, karena operand maka 3 dijadikan hasil saja

5 \* ( 2 + 3 ) / 1 – 6

+(

\*

5 2 3

1. Lanjut ke operator “)”, operator ini akan mengeluarkan seluruh isi stack dimulai dari top stack hingga bertemu operator “(” yang jadi pasangannya. Karena diantara operator “(” dan “)” hanya ada “+” maka hanya operator “+” saja yang dijadikan hasil.Tanda kurung tidak usah dimasukkan



5 \* (2 + 3 ) / 1 – 6

\*

5 2 3 +

1. Lanjutkan ke operator “/“, operator ini akan dimasukkan ke dalam *stack*. Karena di *top stack* sudah ada isinya, maka bandingkan keduanya. Bila yang akan masuk memiliki derajat yang lebih besar, maka tumpuk saja. Sebaliknya, bila yang akan masuk memiliki derajat yang sama atau lebih kecil, maka keluarkan *top stack* hingga operator yang berada di *top stack* berderajat lebih kecil dari operator yang akan masuk.

Karena “/” berderajat sama dengan “\*” maka keluarkan *top stack* (“\*”). Karena *stack* sudah hampa, maka operator “/” dimasukkan ke dalam *stack* sebagai *top stack*nya.

 5 \* (2 + 3 ) / 1 – 6

/

5 2 3 + \*

1. Lanjutkan ke *operand* “1”, karena sebagai *operand*, maka “1” dijadikan hasil saja.

5 \* (2 + 3 ) / 1 – 6

/

5 2 3 + \* 1

1. Lanjutkan ke operator “-“, operator ini akan dimasukkan ke dalam *stack*. Karena di *top stack* sudah ada isinya, maka bandingkan keduanya. Karena “-“ berderajat lebih kecil dari “/” maka operator “/” dikeluarkan dari tumpukan dan dijadikan hasil.

5 \* (2 + 3 ) / 1 – 6

-

5 2 3 + \* 1 /

1. Lanjutkan ke *operand* “6”, karena sebagai *operand*, maka “6” dijadikan hasil saja.



5 \* (2 + 3 ) / 1 – 6

-

5 2 3 + \* 1 / 6

1. Karena proses telah selesai, maka keluarkan seluruh isi stack mengikuti kaidahnya, *last in first out*. Karena hanya ada “-“ maka hasil akhirnya menjadi:

5 2 3 + \* 1 / 6 -

* Cara binary tree

5 \* (2 + 3 ) / 1 – 6

1. Langkah pertama adalah melihat terlebih dahulu derajat operasi dari notasi infix yang akan diproses, setelahnya disederhanakan menjadi inisial huruf atau angka. Diperoleh ( 2 + 3) lalu anggap saja menjadi A, maka notasi infix menjadi

5 \* A / 1 – 6

1. Setelah itu sederhanakan pula 5 \* A menjadi I, maka notasi infix menjadi

I / 1 - 6

1. Bedasarkan notasi infix di langkah kedua maka sederhanakan lagi I / 1 menjadi J, maka notasi infix menjadi J – 6.

Setelah selesai proses diatas maka selanjutnya kita ubah notasi infix menjadi postfix menggunakan tree. Menurut langkah ketiga yaitu J – 6 ,maka struktur pohon binernya adalah

Karena J merupakan hasil penyederhana dari operasi I / 1 maka struktur pohon binernya adalah

setelah itu gabungkan dengan struktur pohon yang sebelumnya menjadi seperti ini

Karena I merupakan hasil penyederhana dari operasi 5 \* A maka struktur pohon binernya adalah

setelah itu gabungkan dengan struktur pohon yang sebelumnya menjadi seperti ini

Karena A juga merupakan hasil penyederhana dari operasi 5 \* A maka struktur final dari pohon binernya adalah

Lalu melakukan tranversal post order agar ketika disusun notasinya menjadi postfix

7

6

5

4

3

2

1

8

1. 5 2
2. 5 2 3
3. 5 2 3 +
4. 5 2 3 + \*
5. 5 2 3 + \* 1
6. 5 2 3 + \* 1 /
7. 5 2 3 + \* 1 / 6
8. 5 2 3 + \* 1 / 6 -

Setelah notasi infix diubah menjadi notasi postfix menggunakan tree, maka proses selanjutnya ialah menghitung notasi tersebut menggunakan **stack**

Notasi postfix yang akan dihitung adalah 5 2 3 + \* 1 / 6 -

Kondisi stack sebelum di isi oleh notasi yang akan dihitung

|  |
| --- |
|  |
|  |
|  |
|  |

1. Karena notasi postfix sudah berbentuk tree, maka dari itu untuk cara membaca nya dibutuhkan metode traversal. Travesal yang akan dipakai adalah postorder, dimana postorder memiliki prinsip left-right-visit. Maka dari itu jika melihat struktur tree yang di atas kita lakukan push ‘5’ pada stack

push

|  |
| --- |
|  |
|  |
|  |
| 5 |

1. Setelah itu kita lihat subtree samping 5 yaitu 2 dan setelah itu 3 dan operator \* . Kita push terlebih dahulu 2 dan 3 pada stack

Aturan perhitungan pada stack adalah jika yang akan di push selanjutnya adalah operator maka lakukan operasi tersebut pada 2 digit terakhir pada stack sesuai dengan operator yang akan dimasukan (push).

|  |
| --- |
|  |
| 3 |
| 2 |
| 5 |

1. Karena operator + akan di push pada stack, dan jika melihat aturan di langkah kedua maka kita lakukan pop untuk 2 digit pada stack

Kondisi stack setelah di pop

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| 3 | pop |  |  |
| 2 | pop |  |  |
| 5 |  |  | 5 |

1. Lakukan operasi + pada 2 dan 3 . Hasil akhirnya ialah 5, setelah itu push kembali ke stack

|  |
| --- |
|  |
|  |
| 5 |
| 5 |

1. Setelah itu liat kembali tree yang sudah dibuat karena prinsip postorder ialah left-right-visit maka yang selanjutnya akan di push adalah operator \* dan karena mengikuti aturan di langkah kedua maka lakukan pop terhadap 2 digit yang ada di stack

Kondisi stack setelah di pop

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
| 5 | pop |  |  |
| 5 | pop |  |  |

1. Lakukan operasi \* pada 5 dan 5. Hasil akhirnya ialah 25, setelah itu push kembali ke stack

|  |
| --- |
|  |
|  |
|  |
| 25 |

1. Setelah itu liat kembali tree diatas, yang harus di push selanjutnya ialah 1

|  |
| --- |
|  |
|  |
| 1 |
| 25 |

1. Setelah itu liat kembali tree yang sudah dibuat karena prinsip postorder ialah left-right-visit maka yang selanjutnya akan di push adalah operator / dan karena mengikuti aturan di langkah kedua maka lakukan pop terhadap 2 digit yang ada di stack

Kondisi stack setelah di pop

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
| 1 | pop |  |  |
| 25 | pop |  |  |

1. Lakukan operasi / pada 25 dan 1. Hasil akhirnya ialah 25, setelah itu push kembali ke stack

|  |
| --- |
|  |
|  |
|  |
| 25 |

1. Setelah itu liat kembali tree diatas, yang harus di push selanjutnya adalah 6

|  |
| --- |
|  |
|  |
| 6 |
| 25 |

1. Setelah itu liat kembali tree yang sudah dibuat karena prinsip postorder ialah left-right-visit maka yang selanjutnya akan di push adalah operator - dan karena mengikuti aturan di langkah kedua maka lakukan pop terhadap 2 digit yang ada di stack

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
| 6 | pop |  |  |
| 25 | pop |  |  |

Kondisi stack setelah di pop

1. Lakukan operasi – pada 25 dan 6. Hasil akhirnya ialah 19 setelah itu push kembali pada stack

|  |
| --- |
|  |
|  |
|  |
| 19 |

Jadi itulah kondisi terakhir dari stack setelah melakukan perhitungan postfix



## Identifikasi Proses

* 1. Membaca inputan dari user

Setelah user input bilangan yang akan dijumlahkan, sistem akan membaca apa yang sudah diinputkan sebelumnya. Setelah itu sistem akan mencoba mengidentifikasi yang di inputkan tersebut merupakan operator atau operand menggunakan stack.

|  |  |
| --- | --- |
| IS | FS |
| Nilai dari variabel masih belum diketahui. | Variabel berisi nilai yang di input oleh user. |

* 1. Mengubah notasi infix menjadi postfix

Dalam proses ini sistem akan mengubah notasi infix (yang sudah di input user) menjadi postfix menggunakan struktur tree. Proses ini bertujuan untuk memudahkan sistem saat di proses perhitungan.

|  |  |
| --- | --- |
| IS | FS |
| Notasi berjenis infix | Notasi berjenis postfix |

* 1. Melakukan perhitungan

Dalam proses ini akan terjadi proses perhitungan sesuai dengan gambaran dari representasi data yang sudah kami gambarkan dibawah

|  |  |
| --- | --- |
| IS | FS |
| Kondisi stack masih kosong (NULL) | Kondisi stack terisi nilai berjenis postfix yang sebelumnya sudah di inputkan oleh user |

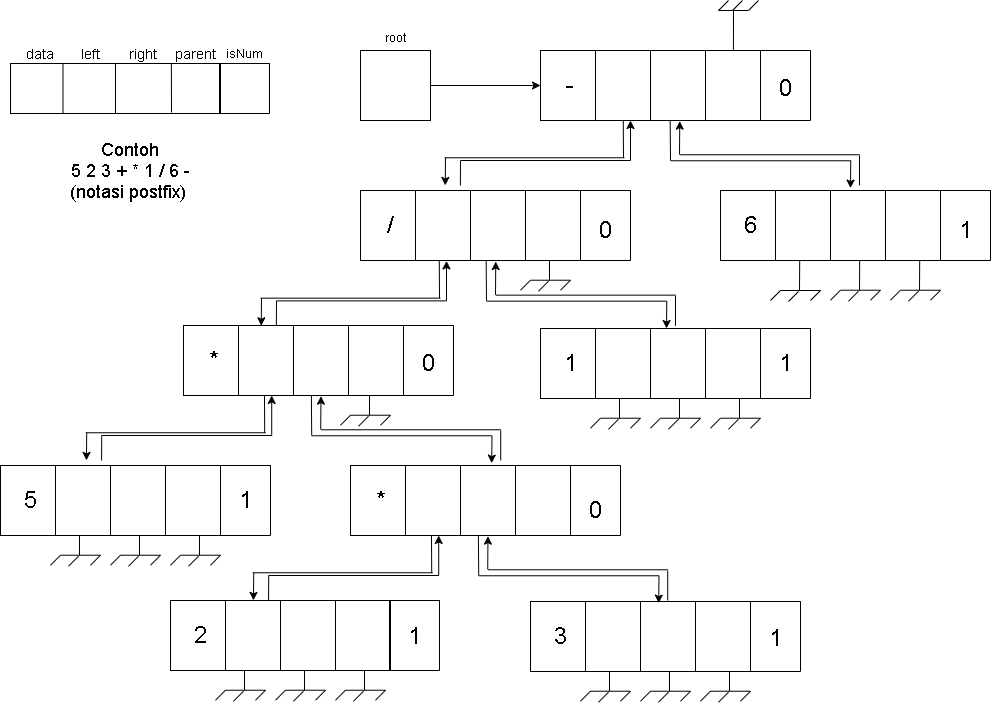
* 1. Menampilkan hasil perhitungan

Setelah melakukkan proses perhitungan, maka akan ada modul untuk

menampilkan hasil dari perhitungan yang sudah di proses menggunakan travesal.

|  |  |
| --- | --- |
| IS | FS |
| 1. Kondisi stack masih kosong (NULL) 2. Notasi belum terhitung | 1. Kondisi stack masih kosong (NULL) 2. Notasi sudah terhitung |

## Identifikasi Data

Representasi struktur data dari proses yang sudah kami buat

Kamus data :

Infotype : Variable komposit dengan subvar sebagai berikut :

* num : variable tunggal bertipe double
* operator : variable tunggal bertipe integer

NodeTree : Variable komposit dengan subvar sebagai berikut :

- IsNum : Variable tunggal bertipe boolean

// variable untuk menyimpan nilai true atau false dari operator yang sudah di input user sebelumnya.

- Info : Variable bertipe Infotype

// variable untuk menyimpan operator atau operand yang sudah di inputkan user

- Left : Variable tunggal bertipe pointer

// variable untuk menyimpan alamat left son dari node

- Right : Variable tunggal bertipe pointer

// variable untuk menyimpan alamat right son dari node

- Parent: Variable tunggal bertipe pointer

// variable untuk menyimpan alamat parent dari node

bTree : variable komposit dengan subvar sebagai berikut:

- root : variable bertipe pointer

StackCalc : Variable komposit dengan subvar sebagai berikut :

* IsNum : variable tunggal bertipe integer
* Info : Variable bertipe Infotype
* Pnode : Variable tunggal bertipe pointer

Stack : Variable komposit dengan subvar sebagai berikut :

* Tnode : Variable tunggal bertipe pointer

Inputinfix : Variable komposit dengan subvar sebagai berikut :

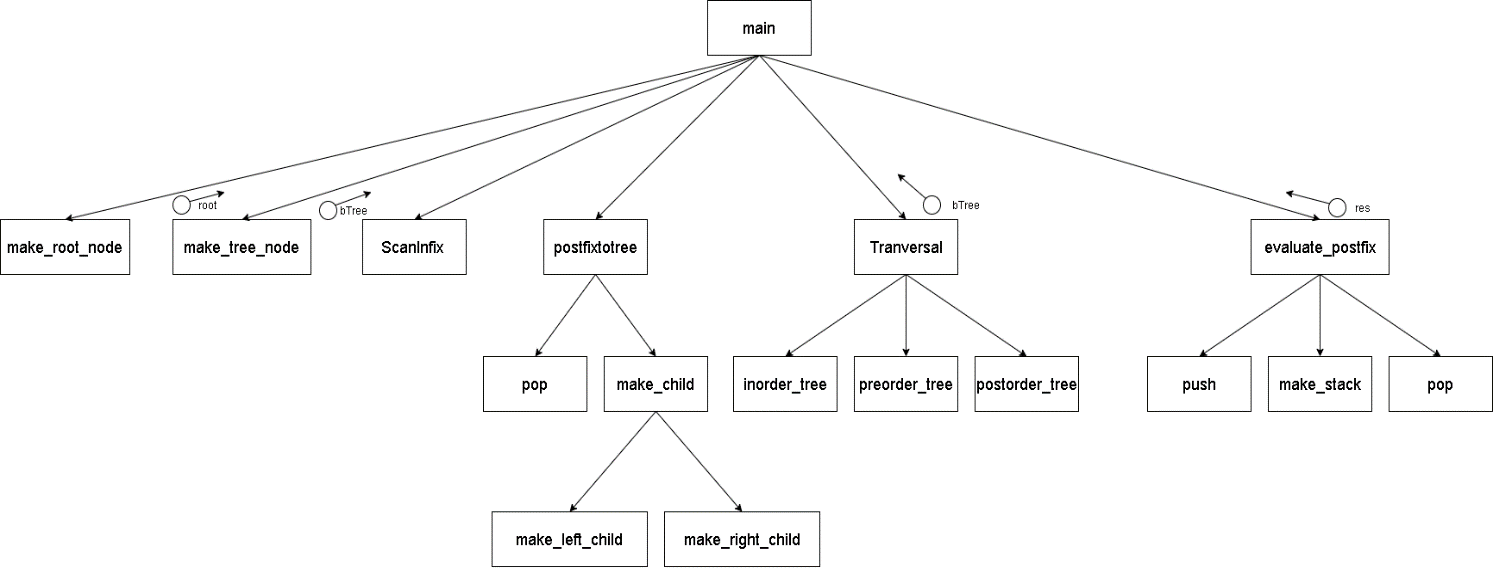
* inputMath : variable tunggal bertipe double
* oprtr : variable tunggal bertipe char

### BAB II

**PERANCANGAN PROGRAM**

## Perancangan proses

# Structured Chart



# Penjelasan Modul

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| No. | Nama Modul | Deskripsi | Jenis | Parameter | Kamus Data (Lokal) |
| 1 | make\_root\_node | Modul ini berfungsi untuk membuat root dari tree | Function | - | - |
| 2 | make\_tree\_node | Modul ini berfungsi untuk membuat struktur tree | Function | - | - |
| 3 | make\_child | Modul ini berfungsi untuk membuat child dari struktur tree yang sudah dibuat | Function | TreeNode \*parent, Infotype, IsNum | - |
| 4 | make\_right\_child | Modul ini merupakan modul untuk membuat right son dari struktur tree yang akan dibuat | Procedure | TreeNode \*parent, Infotype, IsNum | - |
| 5 | make\_left\_child | Modul ini merupakan modul untuk membuat left son dari struktur tree yang akan dibuat | Procedure | TreeNode \*parent, Infotype, IsNum | - |
| 6 | traverseal\_tree | Modul ini merupakan modul untuk melakukan traversal pada tree | Procedure | bTree\*root, int method | - |
| 7 | preorder\_tree | Modul ini merupakan modul untuk melakukan traversal preorder pada tree. | Procedure | bTree\*traverNode | - |
| 8 | inorder\_tree | Modul ini merupakan modul untuk melakukan traversal inorder pada tree. | Procedure | bTree\*traverNode | - |
| 9 | postorder\_tree | Modul ini merupakan modul untuk melakukan traversal postorder pada tree. | Procedure | bTree\*traverNode | - |
| 10 | make\_stack | Modul ini merupakan modul untuk mengalokasikan Tnode dari stack | Function | - |  |
| 11 | push | Modul ini merupakan modul untuk insert node baru ke stack | Procedure | stack ,  Infotype. IsNum |  |
| 12 | pop | Modul ini merupakan modul ntuk mendelete node pada stack | Function | stack |  |
| 13 | evaluate\_postfix | Modul ini merupakan modul untuk menghitung notasi postfix | Function | char\* res |  |

# Algoritma Proses

**make\_root\_node()🡪 bTree**

Begin

Alokasi temp

If (Not temp)

Then write (layar) “Memori sudah terisi!.”

Endif

End modul make\_root\_node

**make\_tree\_node()🡪 NodeTree**

Begin

Alokasi temp

If (Not temp)

Then write (layar) “Memori sudah terisi!.”

Endif

End modul make\_tree\_node

**make\_stack()🡪 Stack**

Begin

Alokasi temp

If (Not temp)

Then write (layar) “Memori sudah terisi!.”

Endif

End modul make\_stack

**Modul evaluate\_postfix**

Begin

Struct stack 🡨 make\_stack(strlen(res))

Int i

If(!stack) return -1

for ( i=0; res[i]; ++i )

if(isdigit(res[i]))

push(stack, res[i] – ‘0’)

else

int val1 🡨 pop(stack)

int val2 🡨 pop(stack)

switch (res[i])

case : ‘+’ : push (stack, val2 + val 1)

break

case : ‘-’ : push (stack, val2 - val 1)

break

case : ‘\*’ : push (stack, val2 \* val 1)

break

case : ‘/’ : push (stack, val2 / val 1)

break

endif

endfor

return pop(stack)

endif

End Modul evaluate\_postfix

### BAB III

**KESIMPULAN**

Program kalkulator yang dibuat akan menggunakan struktur data list stack dan struktur data tree, serta menggunakan proses prefix, infix dan postfix. Konsep dari program ini adalah mengubah infix menjadi postfix menggunakan **stack**, setelah itu menggunakan struktur data tree untuk notasi aritmatika. Dalam program kalkulator yang akan kami buat kira kira kami akan membuat fitur penjumlahan, penguraangan, perkalian dan pembagian.

# DAFTAR PUSTAKA

# DAFTAR KONTRIBUSI ANGGOTA KELOMPOK

|  |  |
| --- | --- |
| No. | : 1 |
| Nama | : Alifah Fisalsabilawati |
| Kontribusi | :   1. Mengerjakan bagian 1.1 Deskripsi program bagian deskripsi 2. Mengerjakan representasi struktur data (gambar) 3. Mengerjakan proses ubah infix ke postfix menggunakan binary tree 4. Melengkapi identifikasi proses 5. Melengkapi kamus data 6. Menambahkan modul 7. Membuat algoritma proses 8. Membuat gambaran proses perhitungan pada stack 9. Melengkapi kamus data |

|  |  |
| --- | --- |
| No. | : 2 |
| Nama | : Faishal Muhammad |
| Kontribusi | :   1. Mengerjakan bagian 1.1 Deskripsi program bagian konsep program 2. Mengerjakan identifikasi proses 3. Membuat kamus data 4. Melengkapi identifikasi proses 5. Melengkapi kamus data 6. Menambahkan modul 7. Membuat structure chart 8. Menambahkan proses penetuan postfix menggunakan traversal postorder 9. Melengkapi kamus data |