Contoh Praktikum Algoritma dan Struktur Data

Nama : Agil Deriansyah Hasan
Nim : 4522210125

Dosen Pengajar :

Dra.SRI REZEKI CANDRA NURSARI,M.Kom
Prak. Algoritma dan Struktur Data - I

S1-Teknik Informatika
Fakultas Teknik
Universitas Pancasila 2023/2024

cnthprak9-1

asd8.cpp | asd9.cpp | cnthprak8-1.cpp | pasd8-2.cpp | cnthprak9-1.cpp

```cpp
#include <iostream>
using namespace std;

typedef struct node *alamatnode;
typedef struct node {
    char info;
    alamatnode right;
    alamatnode left;
} NODE;

typedef struct {
    NODE* root;
} TREE;

void BuatTree(char C, TREE *T) {
    NODE *baru = new NODE;
    baru->info = C;
    baru->right = NULL;
    baru->left = NULL;
    T->root = baru;
}

void TambahKanan(char C, NODE *root) {
    if (root->right == NULL) {
        NODE *baru = new NODE;
        baru->info = C;
        baru->right = NULL;
        baru->left = NULL;
        root->right = baru;
    } else {
        cout << "Sub Tree Kanan telah disisi" << endl;
    }
}

void TambahKiri(char C, NODE *root) {
    if (root->left == NULL) {
        NODE *baru = new NODE;
        baru->info = C;
        baru->right = NULL;
        baru->left = NULL;
        root->left = baru;
    } else {
        cout << "Sub Tree Kiri telah disisi" << endl;
    }
}

void HapusAll(NODE *root) {
    if (root != NULL) {
        HapusAll(root->left);
        HapusAll(root->right);
        delete root;
```

asd8.cpp | asd9.cpp | cnthprak8-1.cpp | pasd8-2.cpp | cnthprak9-1.cpp

```cpp

void HapusAll(NODE *root) {
    if (root != NULL) {
        HapusAll(root->left);
        HapusAll(root->right);
        delete root;
    }
}

void HapusKanan(NODE *root) {
    if (root != NULL && root->right != NULL) {
        HapusAll(root->right);
        root->right = NULL;
    }
}

void HapusKiri(NODE *root) {
    if (root != NULL && root->left != NULL) {
        HapusAll(root->left);
        root->left = NULL;
    }
}

void CetakTreePreOrder(NODE *root) {
    if (root != NULL) {
        cout << root->info << " ";
        CetakTreePreOrder(root->left);
        CetakTreePreOrder(root->right);
    }
}

void CetakTreeInOrder(NODE *root) {
    if (root != NULL) {
        CetakTreeInOrder(root->left);
        cout << root->info << " ";
        CetakTreeInOrder(root->right);
    }
}

void CetakTreePostOrder(NODE *root) {
    if (root != NULL) {
        CetakTreePostOrder(root->left);
        CetakTreePostOrder(root->right);
        cout << root->info << " ";
    }
}
```

asd8.cpp | asd9.cpp | cnthprak8-1.cpp | pasd8-2.cpp | cnthprak9-1.cpp

```
92
93  void CopyTree(NODE *root1, NODE **root2) {
94      if (root1 != NULL) {
95          *root2 = new NODE;
96          (*root2)->info = root1->info;
97          (*root2)->left = NULL;
98          (*root2)->right = NULL;
99          CopyTree(root1->left, &(*root2)->left);
100         CopyTree(root1->right, &(*root2)->right);
101     }
102 }
103
104 bool isEqual(NODE *root1, NODE *root2) {
105     if (root1 == NULL && root2 == NULL) {
106         return true;
107     }
108     if (root1 == NULL || root2 == NULL) {
109         return false;
110     }
111     return (root1->info == root2->info &&
112             isEqual(root1->left, root2->left) &&
113             isEqual(root1->right, root2->right));
114 }
115
116 int main() {
117     TREE T;
118     BuatTree('R', &T);
119     TambahKiri('S', T.root);
120     TambahKanan('U', T.root);
121     TambahKiri('V', T.root->left);
122     TambahKanan('W', T.root->left);
123     TambahKiri('Y', T.root->right);
124     TambahKanan('Z', T.root->right);
125
```

asd8.cpp | asd9.cpp | cnthprak8-1.cpp | pasd8-2.cpp | cnthprak9-1.cpp

```
116 int main() {
117     TREE T;
118     BuatTree('R', &T);
119     TambahKiri('S', T.root);
120     TambahKanan('U', T.root);
121     TambahKiri('V', T.root->left);
122     TambahKanan('W', T.root->left);
123     TambahKiri('Y', T.root->right);
124     TambahKanan('Z', T.root->right);
125
126     cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;
127     cout << "~~~~~~~  PreOrder  ~~~~~~" << endl;
128     cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;
129     cout << endl;
130     CetakTreePreOrder(T.root);
131     cout << endl;
132     cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;
133     cout << "~~~~~~~  InOrder  ~~~~~~~" << endl;
134     cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;
135     cout << endl;
136     CetakTreeInOrder(T.root);
137     cout << endl;
138     cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;
139     cout << "~~~~~~~ PostOrder ~~~~~~~" << endl;
140     cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;
141     cout << endl;
142     CetakTreePostOrder(T.root);
143     cout << endl;
144
145     TREE T2;
146     CopyTree(T.root, &T2.root);
147
148     if (isEqual(T.root, T2.root)) {
149         cout << "Tree yang sama" << endl;
150     } else {
151         cout << "Tree yang tidak sama" << endl;
152     }
153
154     HapusKanan(T.root->left);
155     HapusKiri(T.root->left);
156     cout << endl;
157     cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;
158     cout << "PreOrder Setelah Dihapus" << endl;
159     cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;
160     cout << endl;
161     CetakTreePreOrder(T.root);
162     cout << endl;
163     cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;
164
165     return 0;
166 }
```


Command Prompt
C:\Users\agild>F:
F:\>g++ cnthprak9-1.cpp -o 1
F:\>1
~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~ PreOrder ~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~
R S V W U Y Z
~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~ InOrder ~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~
V S W R Y U Z
~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~ PostOrder ~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~
V W S Y Z U R
Tree yang sama
~~~~~~~~~~~~~~~~~~~~~~~~~
PreOrder Setelah Dihapus
~~~~~~~~~~~~~~~~~~~~~~~~~
R S U Y Z
~~~~~~~~~~~~~~~~~~~~~~~~~
F:\>

Pseudocode :

```
Kamus/Deklarasi Variabel fungsi BuatTree
C = char

Algoritma/Deskripsi fungsi BuatTree(char C, TREE *T)
NODE *baru = new NODE
baru->info = C
baru->right = NULL
baru->left = NULL
T->root = baru

Kamus/Deklarasi Variabel fungsi TambahKanan
C = char

Algoritma/Deskripsi fungsi TambahKanan( C, NODE *root)
if (root->right == NULL)
        NODE *baru = new NODE
        baru->info = C
        baru->right = NULL
        baru->left = NULL
        root->right = baru
else
        print ("Sub Tree Kanan telah disisi")
endif

Kamus/Deklarasi Variabel fungsi TambahKiri
C = char

Algoritma/Deskripsi fungsi (C, NODE *root)
if (root->left == NULL)
        NODE *baru = new NODE
        baru->info = C
        baru->right = NULL
        baru->left = NULL
        root->left = baru
else
        print ("Sub Tree Kiri telah disisi")
endif
```

```
Kamus/Deklarasi Variabel fungsi HapusAll
-

Algoritma/Deskripsi fungsi HapusAll(NODE *root)
if (root != NULL)
        HapusAll(root->left)
        HapusAll(root->right)
        delete root
endif

Kamus/Deklarasi Variabel fungsi HapusKanan
-

Algoritma/Deskripsi fungsi HapusKanan(NODE *root)
if (root != NULL && root->right != NULL)
        HapusAll(root->right)
        root->right = NULL

Kamus/Deklarasi Variabel fungsi HapusKiri
-

Algoritma/Deskripsi fungsi HapusKiri(NODE *root)
if (root != NULL && root->left != NULL)
        HapusAll(root->left)
        root->left = NULL

Kamus/Deklarasi Variabel fungsi CetakTreePreOrder
-

Algoritma/Deskripsi fungsi CetakTreePreOrder(NODE *root)
if (root != NULL)
      print root->info
      CetakTreePreOrder(root->left)
      CetakTreePreOrder(root->right)
```

```
Kamus/Deklarasi Variabel fungsi CetakTreeInOrder
-

Algoritma/Deskripsi fungsi CetakTreeInOrder(NODE *root)
if (root != NULL)
        CetakTreeInOrder(root->left)
        pritn root->info
        CetakTreeInOrder(root->right)
endif

Kamus/Deklarasi Variabel fungsi CetakTreePostOrder
-

Algoritma/Deskrispi fungsi CetakTreePostOrder(NODE *root)
if (root != NULL)
        CetakTreePostOrder(root->left)
        CetakTreePostOrder(root->right)
        print root->info
endif

Kamus/Deklarasi Variabel fungsi CopyTree
-

Algoritma/Deskripsi fungsi CopyTree(NODE*root1,
NODE**root2)
if (root1 != NULL)
        *root2 = new NODE
        (*root2)->info = root1->info
        (*root2)->left = NULL
        (*root2)->right = NULL
        CopyTree(root1->left, &(*root2)->left)
        CopyTree(root1->right, &(*root2)->right)
endif
```

```
Kamus/Deklarasi Variabel fungsi isEqual
-

Algoritma/Deskripsi fungsi isEqual(NODE *root1,
NODE *root2)
if (root1 == NULL && root2 == NULL)
        return true
endif
if (root1 == NULL || root2 == NULL)
        return false
endif
return (root1->info == root2->info &&
         isEqual(root1->left, root2->left) &&
         isEqual(root1->right, root2->right))
```

```
Kamus/Deklarasi Variabel fungsi utama
info = char

Algoritma/Deskripsi fungsi utama
struct node *alamatnode
struct node { info , right , left }
node NODE
struct { root }
TREE
TREE T
BuatTree('R', &T)
TambahKiri('S', T.root)
TambahKanan('U', T.root)
TambahKiri('V', T.root->left)
TambahKanan('W', T.root->left)
TambahKiri('Y', T.root->right)
TambahKanan('Z', T.root->right)
CetakTreePreOrder(T.root)
CetakTreeInOrder(T.root)
CetakTreePostOrder(T.root)
TREE T2
CopyTree(T.root, &T2.root)
if (isEqual(T.root, T2.root))
        "Tree yang sama"
else
        "Tree yang tidak sama"
endif
HapusKanan(T.root->left)
HapusKiri(T.root->left)
CetakTreePreOrder(T.root)
return 0
```

Algoritma :

1. Membuat fungsi BuatTree(C,Tree *T)
2. NODE *baru = new NODE
3. baru->info = C
4. baru->right = NULL
5. baru->left = NULL
6. T->root = baru
7. Membuat fungsi TambahKanan(C, NODE *root)
8. Jika (root -> right == NULL) maka kerjakan baris 9 s.d 13 kalau tidak baris 14
9. NODE *baru = new NODE
10. baru->info = C
11. baru->right = NULL
12. baru->left = NULL
13. root->right = baru
14. Mencetak/Menampilkan "Sub Tree Kanan telah diisi"
15. Membuat fungsi TambahKiri (C, NODE *root)
16. Jika (root -> left == NULL) maka kerjakan baris 17 s.d 21 kalau tidak kerjakan baris 22
17. NODE *baru = new NODE
18. baru->info = C
19. baru->right = NULL
20. baru->left = NULL
21. root->left = baru
22. Mencetak/Menampilkan "Sub Tree Kiri telah disisi"
23. Membuat fungsi HapusAll (NODE *root)
24. Jika (root != NULL) maka kerjakan baris 25 s.d 27
25. HapusAll(root->left)
26. HapusAll(root->right)
27. Menghapus root
28. Membuat fungsi HapusKanan(NODE *root)
29. Jika (root != NULL && root->right != NULL) maka kerjakan baris 30 s.d 31
30. HapusAll(root->right)
31. root->right = NULL
32. Membuat fungsi HapusKiri(NODE *root)
33. Jika (root != NULL && root->left != NULL)
34. HapusAll(root->left)
35. root->left = NULL
36. Membuat fungsi CetakTreePreOrder(NODE *root)
37. Jika (root != NULL) maka kerjakan baris 38 s.d 40
38. Mencetak/Menamplkan Nilai root -> info
39. Memanggil fungsi CetakTreePreOrder(root->left)
40. Memanggil fungsi CetakTreePreOrder(root->right)
41. Membuat fungsi CetakTreeInOrder(NODE *root)
42. Jika (root != NULL) maka kerjakan baris 43 s.d 45
43. Memanggil fungsi CetakTreeInOrder(root -> left)
44. Mencetak/Menampilkan Nilai root -> info
45. Memanggil fungsi CetakTreeInOrder(root -> right)
46. Membuat fungsi CetakTreePostOrder (NODE *root)
47. Jika (root != NULL) Maka kerjakan baris 48 s.d 50
48. CetakTreePostOrder(root->left)
49. CetakTreePostOrder(root->right)
50. Mencetak/Menampilkan Nilai root -> info
51. Membuat fungsi CopyTree(NODE*root1,NODE**root2)
52. Jika (root1 != NULL) maka kerjakan baris 53 s.d
53. *root2 = new NODE
54. (*root2)->info = root1->info
55. (*root2)->left = NULL
56. (*root2)->right = NULL
57. Memanggil fungsi CopyTree(root1->left,&(*root2)->left)
58.MemanggilfungsiCopyTree(root1->right,&(*root2)->right
59. Membuat fungsi IsEqual(NODE *root1, NODE *root2)
60. Jika (root1 == NULL && root2 == NULL) maka kerjakan baris 61
61. return true
62. Jika (root1 == NULL || root2 == NULL) maka kerjakan baris
63. return false
64. return (root1->info == root2->info &&
    isEqual(root1->left, root2->left) &&
    isEqual(root1->right, root2->right))

65. Membuat fungsi utama
66. Deklarasi struktur(struct node *alamatnode
66. Deklarasi struktur(struct node {info , alamatnode right, alamat left})
67. Mendefinisikan struktur (NODE)
68. Membuat struktur(struct {NODE* root})
69. Mendefinisikan struktur (TREE)
70. TREE T
71. BuatTree('R', &T)
72. TambahKiri('S', T.root)
73. TambahKanan('U', T.root)
74. TambahKiri('V', T.root->left)
75. TambahKanan('W', T.root->left)
76. TambahKiri('Y', T.root->right)
77. TambahKanan('Z', T.root->right)
78. CetakTreePreOrder(T.root)
79. CetakTreeInOrder(T.root)
80. CetakTreePostOrder(T.root)
81. TREE T2
82. Memanggil fungsi CopyTree(T.root, &T2.root)
83. Jika (isEqual(T.root, T2.root)) maka kerjakan baris 84 kalau tidak baris 85
84. Mencetak/Menampilkan “Tree yang sama”
85. Mencetak/Menampilkan “Tree yang tidak sama”
86. HapusKanan(T.root->left)
87. HapusKiri(T.root->left)
88. CetakTreePreOrder(T.root)
89. Selesai

cnthprak9-2

asd8.cpp | asd9.cpp | cnthprak8-1.cpp | pasd8-2.cpp | asd10.cpp

```cpp
#include <iostream>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
using namespace std;

struct Node{
  char info;
  struct Node* Left;
  struct Node* Right;
};

typedef struct Node simpul;
simpul *root, *P, *Q[30], *R, *current;
char X;

void inisialisasi(){
  root = NULL;
  P = NULL;
}

void buatsimpul(char X){
  P = (simpul*) malloc(sizeof(simpul));
  if(P != NULL){
    P->info = X;
    P->Left = NULL;
    P->Right = NULL;
  }
}

void buatsimpulakar(){
  if(root == NULL){
    if(P != NULL){
      root = P;
      P->Left = NULL;
      P->Right = NULL;
    }
    else{
      cout << "Simpul belum dibuat" << endl;
    }
  }
  else{
    cout << "Pohon Sudah Ada !!!" << endl;
  }
}
```

asd8.cpp | asd9.cpp | cnthprak8-1.cpp | pasd8-2.cpp | asd10.cpp

```cpp

void innserturutnomer(char input[6]){
  int i, j, flag;
  char X;
  flag = 0;
  i = 1;
  j = 1;
  Q[i] = root;
  while(flag == 0 && j < 6){
    X = input[j-1];
    if(X != '0'){
      buatsimpul(X);
      current = Q[i];
      current->Left = P;
      j++;
      Q[j] = P;
    }
    else{
      flag = 1;
      j++;
      Q[j] = NULL;
    }
    if(flag == 0){
      X = input[j-1];
      if(X != '0'){
        buatsimpul(X);
        current->Right = P;
        j++;
        Q[j] = P;
      }
    }
    else{
      flag = 1;
      j++;
      Q[j] = NULL;
    }
    i++;
  }
}
```

asd8.cpp | asd9.cpp | cnthprak8-1.cpp | pasd8-2.cpp | asd10.cpp

```cpp
void bacaurutnomer(){
  int i, j, n, counter;
  i = 1;
  j = 1;
  n = 1;
  counter = 0;
  int level = 0;
  while (Q[i] != NULL){
    current = Q[i];
    if(i == 1){
      cout << endl;
      cout << "Level " << level << endl;
    }
    cout<<current -> info <<" - ";
    counter++;
    if(counter == n){
        level++;
        cout<<"Level " << level <<" " <<endl;
    }
    if(counter == n){
      cout << endl;
      counter = 0;
      n = n * 2;
    }
    if(current->Left != NULL){
      j++;
      Q[i] = current->Left;
    }
    if(current->Right != NULL){
      j++;
      Q[i] = current->Right;
    }
    i++;
  }
}

int main(){
  char root = 'R';
  char daun[6] = {'S', 'U', 'V', 'W', 'Y', 'Z'};
  inisialisasi();
  buatsimpul(root);
  buatsimpulakar();
  innserturutnomer(daun);
  bacaurutnomer();
  return 0;
}
```

Command Prompt

```
F:\>g++ asd10.cpp -o 1

F:\>1

Level 0
R - Level 1

S - U - Level 2

V - W - Y - Z - Level 3


F:\>
```

Pseudocode

```
Kamus/Deklarasi Variabel fungsi inisialisasi
-

Algoritma/Deskripsi fungsi inisialisasi
root = NULL
P = NULL

Kamus/Deklarasi Variabel fungsi buatsimpul
X = char

Algoritma/Deskripsi fungsi buatsimpul (X)
P = (simpul*) malloc(sizeof(simpul))
if(P != NULL)
        P->info = X
        P->Left = NULL
        P->Right = NULL
endif

Kamus/Deklarasi Variabel fungsi buatsimpulakar
-

Algoritma/Deskripsi fungsi buatsimpulakar
if(root == NULL)
        if(P != NULL)
                root = P
                P->Left = NULL
                P->Right = NULL
                else
                        print("Simpul belum dibuat")
        endif
```

```
Kamus/Deklarasi Variabel fungsi inserturutnomer
input[6], X = char
i, j, flaq = int

Algoritma/Deskripsi fungsi inserturutnomer(input [6])
flag = 0
i = 1
j = 1
Q[i] = root
while(flag == 0 && j < 6)
        X = input[j-1]
        if(X != '0')
                buatsimpul(X)
                current = Q[i]
                current->Left = P
                j++
                Q[j] = P
        else
                flag = 1
                j++
                Q[j] = NULL
        endif
        if(flag == 0)
                X = input[j-1]
                if(X != '0')
                        buatsimpul(X)
                        current->Right = P
                        j++
                        Q[j] = P
                else
                        flag = 1
                        j++
                        Q[j] = NULL
        endif
i++
endwhile
```

```
Kamus/Deklarasi Variabel fungsi bacaurutnomer
i, j, n, counter, level = int

Algoritma/Deskripsi fungsi bacaurutnomer
i = 1
j = 1
n = 1
counter = 0
level = 0
while (Q[i] != NULL)
        current = Q[i]
        if(i == 1)
                print level
                print current -> info
                counter++
        endif
                if(counter == n)
                        level++
                        print leve
                endifl
                        if(counter == n)
                                counter = 0
                                n = n * 2
                        endif
                                if(current->Left != NULL)
                                        j++
                                        Q[i] = current->Left
                                endif
                                if(current->Right != NULL)
                                j++
                                Q[i] = current->Right
                                endif
i++
endif
endwhile
```

```
Kamus/Deklarasi Variabel fungsi utama
root, daun[6] = char
info = char
X = char

Algoritma/Deskripsi fungsi utama
struct Node{ info , left , right}
typedef struct Node simpul { simpul *root, *P, *Q[30], *R, *current, X}
root = 'R'
char daun[6] = {'S', 'U', 'V', 'W', 'Y', 'Z'}
nisialisasi()
buatsimpul(root)
buatsimpulakar()
innserturutnomer(daun)
bacaurutnomer()
```

Algoritma :

1. Membuat fungsi inisialisasi
2. root = NULL
3. P = NULL
4. Membuat fungsi buatsimpul(X)
5. P = (simpul*) malloc(sizeof(simpul));
6. Jika (P != NULL) maka kerjakan baris 7 s.d 9
7. P->info = X
8. P->Left = NULL
9. P->Right = NULL
10. Membuat fungsi buatsimpulakar
11. Jika (root == NULL) maka kerjakan baris 12 s.d 16 kalau tidak baris 17
12. Jika (P != NULL) maka kerjakana baris 13 s.d 15 kalau tidak baris 16
13. root = P
14. P->Left = NULL
15. P->Right = NULL
16. Mencetak "Simpul belum dibuat"
17. Mencetak "Pohon Sudah Ada !!!"
18. Membuat fungsi inserturutnomer(input [6])
19. flaq = 0
20. i = 1
21. j = 1
22. Q[i] = root
23. Selama (flag == 0 && j < 6) maka kerjakan baris 24 s.d 44
24. X = input[j-1]
25. Jika (X != '0') maka kerjakan baris 26 s.d 30 kalau tidak baris 31 s.d 33
26. Memanggil fungsi buatsimpul(X)
27. current = Q[i]
28. current->Left = P
29. j++
30. Q[j] = P
31. flag = 1
32. j++
33. Q[j] = NULL
34. Jika (flaq == 0) maka kerjakan baris 35 s.d 40
35. X = input[j-1]
36. Jika (X != '0') maka kerjakan baris 37 s.d 40 kalau tidak baris 41 s.d 43
37. buatsimpul(X)
38. current->Right = P
39. j++
40. Q[j] = P
41. flag = 1
42. j++
43. Q[j] = NULL
44. i++
45. Membuat fungsi bacaurutnomer
46. i = 1
47. j = 1
48. n = 1
49. counter = 0
50. level = 0
51. Selama (Q[i] != NULL) maka kerjakan baris 52 s.d
52. current = Q[i]
53. Jika (i == 1) maka kerjakan baris 54
54. Mencetak/Menampilkan Nilai level
55. Mencetak/Menampilkan Nilai current -> info
56. counter ++
57. Jika (counter == n) maka kerjakan baris 58 s.d 59
58. level ++
59. Mencetak/Menampilkan nilai level
60. Jika (counter == n) maka kerjakan baris 61 s.d 62
61. counter = 0
62. n = n * 2
63. Jika (current->Left != NULL) maka kerjakan baris 64 s.d 65
64. j++
65. Q[i] = current->Left
67. Jika (current->Right != NULL) maka kerjakan baris 68 s.d 69
68. j++
69. Q[i] = current->Right
70. i++

71. Membuat fungsi utama
72. Deklarasi struktur (struct Node{info , left,right})
73. Deklarasi typedef struktur (struct Node simpul {simpul *root, *P, *Q[30], *R, *current, X}
74. root = 'R'
75. daun[6] = {'S', 'U', 'V', 'W', 'Y', 'Z'}
76. inisialisasi()
77. buatsimpul(root)
78. buatsimpulakar()
79. innserturutnomer(daun)
80. bacaurutnomer()
81. Selesai