TUGAS PERTEMUAN 7

Nama : Zaldy Seno Yudhanto

Kelas : 2C – Informatika

NPM : 2310631170123

1. Buatlah laporan dari source code pada link di bawah ini :
(Mengukur Maximal Depth Pada Binary Tree CPP Program)

Jawab!

```cpp
#include <iostream>
#include <string>

using namespace std;

struct Node {
    char label;
    Node *right, *left, *parent;
};

Node *root = NULL;

bool emptyTree() {
    return root == NULL;
}

```

= Yang pertama dilakukan adalah membuat sebuah **struct** yang diberi nama *Node,* lalu didalam struct tersebut terdapat sebuah tipe data char yang berguna untuk menyimpan sebuah label lalu ada simpul untuk menghubungkan setiap label yang dibuat agar menjadi tree.

Kemudian didefinisikan isi dari root adalah kosong, kemudian dibuat sebuah fungsi pengembalian dimana akan mengembalikan root menjadi kosong.

```cpp
void createTree(char label) {
    if (root) {
        cout << "Tree sudah dibuat" << endl;
    } else {
        root = new Node();
        root->label = label;
        root->left = NULL;
        root->right = NULL;
        root->parent = NULL;
        cout << label << " Berhasil menjadi root" << endl;
    }
}
```

Kemudian dibuat fungsi kosong yang diberi nama *createTree* yang digunakan untuk membuat sebuah pohon dengan menggunakan linkedlist. Disitu ada pengkondisian dimana jika *root* sudah ada maka akan diberi pesan bahwa ”Tree sudah dibuat”, namun jika belum ada maka root akan didefinisikan sebagai node baru dan *left, right,* dan *parrentnya* adalah kosong sehingga kita bisa mengisinya dengan data yang kita inginkan. Kemudian setelah terbuat simpulnya akan diberi pesan bahwa simpul berhasil dibuat.

Kemudian dibuat sebuah fungsi yang digunakan untuk mencari sebuah label yang sudah dibuat, disana ada pengkondisian dimana, jika node yang dicari tidak ada maka akan mengembalikan NULL, dan jika yang dicari ada maka akan akan diberikan label yang dicari tadi.

```
30  Node* search(Node* node, char label) {
31      if (!node) {
32          return NULL;
33      }
34      if (node->label == label) {
35          return node;
36      }
37      Node* leftSearch = search(node->left, label);
38      if (leftSearch) {
39          return leftSearch;
40      }
41      return search(node->right, label);
42  }
```

```
44  Node* insert(char label, char parentLabel, string child) {
45      if (emptyTree()) {
46          cout << "Tree masih kosong, Tolong dibuat dulu!" << endl;
47          return NULL;
48      }
49
50      Node* parentNode = search(root, parentLabel);
51      if (!parentNode) {
52          cout << "Parent node tidak ditemukan!" << endl;
53          return NULL;
54      }
55
56      if (child == "left") {
57          if (parentNode->left) {
58              cout << "Anak bagian kiri dari node " << parentNode->label << " sudah ada isinya!" << endl;
59          } else {
60              Node* newNode = new Node();
61              newNode->label = label;
62              newNode->left = NULL;
63              newNode->right = NULL;
64              newNode->parent = parentNode;
65              parentNode->left = newNode;
66              cout << "Label " << label << " berhasil dibuat di anak kiri dari node " << parentNode->label << endl;
67              return newNode;
68          }
```

```
    } else if (child == "right") {
        if (parentNode->right) {
            cout << "Anak bagian kanan dari node " << parentNode->label << " sudah ada isinya!" << endl;
        } else {
            Node* newNode = new Node();
            newNode->label = label;
            newNode->left = NULL;
            newNode->right = NULL;
            newNode->parent = parentNode;
            parentNode->right = newNode;
            cout << "Label " << label << " berhasil dibuat di anak kanan dari node " << parentNode->label << endl;
            return newNode;
        }
    }
    return NULL;
}
```

Kemudian dibuat sebuah fungsi yang digunakan untuk memasukan sebuah label, lalu disana ada pengkondisian dimana jika masih belum ada isinya akan diberi pesan bahwa tree masih kosong.

Lalu disana dicari dulu parent nodenya apakah ada, jika parentnya tidak ada maka akan diberi pesan bahwa parent yang diinginkan tidak ada, namun jika parent ditemukan maka akan lanjut pengkondisian selanjutnya, dimana kita akan diminta untuk memasukan labelnya dikiri atau dikanan *parent.* Dimana jika kiri atau kanan parent sudah ada isinya maka akan diberi pesan bahwa tempat ini sudah ada isinya, namun jika masih kosong maka akan dibuat simpul baru dan kemudian simpil baru itu akan dijadikan sebagai simpul kiri atau kanan seperti yang kita minta tadi saat membuat sebuah label.

```
86  void updateLabel(char label, Node* node) {
87      if (emptyTree()) {
88          cout << "Buat tree terlebih dahulu!" << endl;
89      } else {
90          if (!node) {
91              cout << "Tidak ada node ini atau masih kosong!" << endl;
92          } else {
93              cout << "Label sebelumnya = " << node->label << endl;
94              node->label = label;
95              cout << "Label setelahnya = " << node->label << endl;
96          }
97      }
98  }
```

Kemudian dibuat sebuah fungsi kosong yang digunakan untuk mengupdaye sebuah label yang kita buat, dan disana juga ada pengkondisian dimana jika label masih kosong maka akan diberi pesan untuk membuat tree terlebih dahulu. Namun juka sudah ada isinya maka akan masuk ke pengkondisian lagi dimana jika simpulnya masih kosong maka akan diberi pesan bahwa node ini masih kosong, lalu jika ada maka akan ditunjukan isi dari nodenya.

```
100 void retrieveLabel(Node* node) {
101     if (!node) {
102         cout << "Tidak ada Node ini atau masih kosong" << endl;
103     } else {
104         cout << "Label dari node ini adalah " << node->label << endl;
105     }
106 }
```

Kemudian dibuat sebuah fungsi kosong yang digunakan untuk Mengecek apakah terdapat sebuah node yang ditunjuk pada tree. Disana ada pengkondisian dimana jika node masih kosnong maka akan diberi pesan bahwa node ini masih kosong namun jika sudah ada isinya maka akan ditunjukan label dari nodenya.

```
108 void deleteSub(Node* node) {
109     if (node != NULL) {
110         if (node != root && node == node->parent->left) {
111             node->parent->left = NULL;
112         }
113         deleteSub(node->left);
114         if (node != root && node == node->parent->right) {
115             node->parent->right = NULL;
116         }
117         deleteSub(node->right);
118         if (node == root) {
119             root = NULL;
120             delete root;
121         } else {
122             delete node;
123         }
124     }
125 }
```

Lalu dibuat sebuah fungsi kosong yang digunakan untuk menghapus salah satu label yang sudah dibuat. Disana ada pengkondisian dimana jika node tidak kosong maka akan dicek yang ingin dihapus simpul yang kiri atau yang kana kemudian dideklarasikan bahwa simpulnya kosong (NULL) agar terlihat seperti dihapus.

Lalu ada pengkondisian lagi dimana jika yang ingin dihapus adalah rootnya maka akan dihapus seluruh nodenya.

```
127 void clearTree(Node* node) {
128     if (node != NULL) {
129         if (node != root && node == node->parent->left) {
130             node->parent->left = NULL;
131         }
132         clearTree(node->left);
133         if (node != root && node == node->parent->right) {
134             node->parent->right = NULL;
135         }
136         clearTree(node->right);
137         if (node == root) {
138             root = NULL;
139             delete root;
140         } else {
141             delete node;
142         }
143     }
144 }
```

Kemudian ada fungsi kosong yang diberi nama *clearTree* yang digunakan untuk mengahpus tree yang sudah kita buat tadi agar saat keluar sudah tidak ada tree yang tersisa. Disana ada pengkonidisan yang sama seperti fungsi sebelumnya yaitu *deleteSub*, kita akan mendeklarasikan seluruh simpulnya menjadi NULL, kemudian kita mengapus rootnya dan juga kita akan mengapus simpulnya agar kosong.

```
146 void preOrder(Node* node) {
147     if (!node) {
148         return;
149     } else {
150         cout << node->label << " ";
151         preOrder(node->left);
152         preOrder(node->right);
153     }
154 }
```

Kemudian disini ada fungsi kosong yang digunakan untuk mengunjungi semua root dengan urutan root, kiri, kanan. Disana ada pengkondisian dimana jika nodenya kosong maka akan kembali ke menu, tidak mengeluarkan apa apa, namun jika ada maka akan ditampilkan apa yang sudah dikunjungi.

```
156 void postOrder(Node* node) {
157     if (!node) {
158         return;
159     } else {
160         postOrder(node->left);
161         postOrder(node->right);
162         cout << node->label << " ";
163     }
164 }
```

Kemudian ada fungsi kosong lagi yang digunakan untuk mengunjungi semua root dengan urutan kiri, kanan lalu ke root. Sama seperti post order, disana ada pengkondisian jika nodenya tidak ada dan jika nodenya ada.

```
166  void inOrder(Node* node) {
167      if (!node) {
168          return;
169      } else {
170          inOrder(node->left);
171          cout << node-> Node* Node::left
172          inOrder(node->right);
173      }
174  }
```

Kemudian dibuat fungsi kosong yang digunakan untuk mengunjungi semua root dengan urutan kiri, kembali ke root, lalu ke kanan. Sama seperti dua fungsi sebelumnya ada pengkondisian yang mengecek terlebih dahulu apakah ada nodenya atau tidak.

```
176  // Function to calculate the maximum depth of the tree
177  int maxDepth(Node* node) {
178      if (node == NULL) {
179          return 0;
180      } else {
181          int leftDepth = maxDepth(node->left);
182          int rightDepth = maxDepth(node->right);
183          return max(leftDepth, rightDepth) + 1;
184      }
185  }
```

Disini ada fungsi untuk menghitung kedalaman maximum dari sebuah tree. Dimana jika nodenya kosong maka akan dikembalikan nilai 0, dan jika ada maka akan dicek node kiri dan kanan lalu ditambah satu, maka itu akan menjadi kedalaman maksimal dari tree yang sudah kita punya.

```
187  int main() {
188      char label, parentLabel;
189      string child;
190      int choice;
191
192      cout << "Masukkan label untuk root: ";
193      cin >> label;
194      createTree(label);
195
196      while (true) {
197          cout << "\n1. Tambah node\n2. Pre-order traversal\n3. In-order traversal\n4. Post-order traversal\n5. Cari node\n6. Hapus subtree\n7. Hitung kedalaman maksimal\n8. Keluar\nPilih: ";
198          cin >> choice;
199
200          switch (choice) {
201              case 1:
202                  cout << "Masukkan label node baru: ";
203                  cin >> label;
204                  cout << "Masukkan label parent node: ";
205                  cin >> parentLabel;
206                  cout << "Masukkan posisi (left/right): ";
207                  cin >> child;
208                  insert(label, parentLabel, child);
209                  break;
```

Kemudian kita masuk ke fungsi utamanya, pertama dideklarasikan terlebih dahulu variabel yang digunakan untuk membuat sebuah tree. Disana ada label, parentLabel dan ada child yang nantinya bisa ditaruh di kiri dan dikanan dari si parentLabel. Kemudian ada variabel choice yang digunakan untuk memilih menu.

Yang pertama kita akan disuruh membuat labelnya terlebih dahulu atau rootnya, kemudian akan disimpan di variabel label dan kemudian memanggil fungsi createTree yang digunakan untuk membuat sebuah tree.

Kemudian ada perulangan yang akan terus berulang sampai kita memilih menu keluar yang akan memberhentikan perulangan tersebut, didalam perulangan terdapat switch case yang digunakan untuk membuat pilihan sebuah menu.

Jika kita memilih (1) maka kita akan diminta untuk membuat label baru yang akan dijadikan sebagai label baru, kemudian kita akan diminta untuk memasukan sebuah parentLabel yang sudah ada tadi, setelah itu kita posisikan mau di kiri dari parent label atau dikanan parent label. Kemudian setelah memilih baru kita panggil fungsi *insert* untuk memasukan data sesuai apa yang sudah kita masukan tadi.

```
210             case 2:
211                 cout << "Pre-order traversal: ";
212                 preOrder(root);
213                 cout << endl;
214                 break;
215             case 3:
216                 cout << "In-order traversal: ";
217                 inOrder(root);
218                 cout << endl;
219                 break;
220             case 4:
221                 cout << "Post-order traversal: ";
222                 postOrder(root);
223                 cout << endl;
224                 break;
225             case 5:
226                 cout << "Masukkan label node yang dicari: ";
227                 cin >> label;
228                 Node* foundNode;
229                 foundNode = search(root, label);
230                 if (foundNode) {
231                     cout << "Node dengan label " << label << " ditemukan." << endl;
232                 } else {
233                     cout << "Node dengan label " << label << " tidak ditemukan." << endl;
234                 }
235                 break;
```

Jika kita memilih (2) maka akan muncul fungsi pre-order yang digunakan untuk memanggil tree sesuai urutan yang yang sudah ditentukan.

Jika kita memilih (3) maka akan muncul fungsi in-order yang digunakan untuk memanggil tree sesuai urutan yang yang sudah ditentukan.

Jika kita memilih (4) maka akan muncul fungsi post-order yang digunakan untuk memanggil tree sesuai urutan yang yang sudah ditentukan.

Jika kita memilih (5) maka kita akan diminta untuk memasukan label node yang ingin dicari, kemudian dideklarasikan sebuah node baru yaitu foundNode yang kita deklarasikan sebagai fungsi *search* yang sudah kita buat di awal tadi. Dimana jika node ditemukan maka akan diberi pesan ditemukan jika tidak maka akan diberi pesan tidak ditemukan.

```
236             case 6:
237                 cout << "Masukkan label node yang akan dihapus subtree-nya: ";
238                 cin >> label;
239                 Node* nodeToDelete;
240                 nodeToDelete = search(root, label);
241                 if (nodeToDelete) {
242                     deleteSub(nodeToDelete);
243                     cout << "Subtree dengan root " << label << " telah dihapus." << endl;
244                 } else {
245                     cout << "Node tidak ditemukan!" << endl;
246                 }
247                 break;
248             case 7:
249                 cout << "Kedalaman maksimal tree: " << maxDepth(root) << endl;
250                 break;
251             case 8:
252                 clearTree(root);
253                 cout << "Program dibuat oleh : Zaldy Seno Yudhanto (2310631170123)\n";
254                 return 0;
255             default:
256                 cout << "Pilihan tidak valid!" << endl;
257                 break;
258         }
259     }
260 }
261
```

Jika kita memilih (6) maka kita akan diminta untuk memasukan label yang akan dihapus, kemudian akan dibuat node baru yang kita deklarasikan sebagai fungsi *search* yang sudah kita buat tadi. Dimana jika label yang dicari ditemukan maka kita akan beri pesan subtree telah terhapus dan jika tidak ditemukan maka akan diberi pesan bahwa node tidak ditemukan.

Jika kita memilih (7) maka akan tampil pesan kedalaman maksimal dari tree yang kita buat.

Jika kita memilih (8) maka akan memanggil funsgi *clearTree* yang digunakan untuk mengapus seluruh tree kemudian kita akan keluar dari perulangan.

Jika kita memilih diluar dari 1-8 maka akan diberi pesan bahwa pilihan tidak valid.

Ini adalah output dari programnya:

```
Masukkan label untuk root: a
a Berhasil menjadi root

1. Tambah node
2. Pre-order traversal
3. In-order traversal
4. Post-order traversal
5. Cari node
6. Hapus subtree
7. Hitung kedalaman maksimal
8. Keluar
Pilih: 1
Masukkan label node baru: b
Masukkan label parent node: a
Masukkan posisi (left/right): left
Label b berhasil dibuat di anak kiri dari node a
```

```
1. Tambah node
2. Pre-order traversal
3. In-order traversal
4. Post-order traversal
5. Cari node
6. Hapus subtree
7. Hitung kedalaman maksimal
8. Keluar
Pilih: 1
Masukkan label node baru: c
Masukkan label parent node: a
Masukkan posisi (left/right): right
Label c berhasil dibuat di anak kanan dari node a
```

Yang pertama disitu saya membuat label awalnya terlebih dahulu dan membuat leafnya yang saya letakan di kiri dan kanan dari label yang saya buat awal.

```
1. Tambah node
2. Pre-order traversal
3. In-order traversal
4. Post-order traversal
5. Cari node
6. Hapus subtree
7. Hitung kedalaman maksimal
8. Keluar
Pilih: 1
Masukkan label node baru: d
Masukkan label parent node: a
Masukkan posisi (left/right): left
Anak bagian kiri dari node a sudah ada isinya!
```

Kemudian disini jika kita memasukan label ke tempat yang sudah ada maka akan muncul pesan seperti itu.

```
1. Tambah node
2. Pre-order traversal
3. In-order traversal
4. Post-order traversal
5. Cari node
6. Hapus subtree
7. Hitung kedalaman maksimal
8. Keluar
Pilih: 1
Masukkan label node baru: d
Masukkan label parent node: b
Masukkan posisi (left/right): left
Label d berhasil dibuat di anak kiri dari node b
```

```
1. Tambah node
2. Pre-order traversal
3. In-order traversal
4. Post-order traversal
5. Cari node
6. Hapus subtree
7. Hitung kedalaman maksimal
8. Keluar
Pilih: 1
Masukkan label node baru: e
Masukkan label parent node: b
Masukkan posisi (left/right): right
Label e berhasil dibuat di anak kanan dari node b
```

```
1. Tambah node
2. Pre-order traversal
3. In-order traversal
4. Post-order traversal
5. Cari node
6. Hapus subtree
7. Hitung kedalaman maksimal
8. Keluar
Pilih: 1
Masukkan label node baru: f
Masukkan label parent node: c
Masukkan posisi (left/right): right
Label f berhasil dibuat di anak kanan dari node c
```

Kemudian saya membuat label baru seperti yang bisa dilihat datas.

```
1. Tambah node
2. Pre-order traversal
3. In-order traversal
4. Post-order traversal
5. Cari node
6. Hapus subtree
7. Hitung kedalaman maksimal
8. Keluar
Pilih: 2
Pre-order traversal: a b d e c f
```

```
1. Tambah node
2. Pre-order traversal
3. In-order traversal
4. Post-order traversal
5. Cari node
6. Hapus subtree
7. Hitung kedalaman maksimal
8. Keluar
Pilih: 3
In-order traversal: d b e a c f
```

```
1. Tambah node
2. Pre-order traversal
3. In-order traversal
4. Post-order traversal
5. Cari node
6. Hapus subtree
7. Hitung kedalaman maksimal
8. Keluar
Pilih: 4
Post-order traversal: d e b f c a
```

Kemudian ini jika kita memilih menu 2,3,dan 4, maka akan ditunjukan semua labelnya sesuai urutan yang sudah ditentukan.

```
1. Tambah node
2. Pre-order traversal
3. In-order traversal
4. Post-order traversal
5. Cari node
6. Hapus subtree
7. Hitung kedalaman maksimal
8. Keluar
Pilih: 5
Masukkan label node yang dicari: c
Node dengan label c ditemukan.
```

```
1. Tambah node
2. Pre-order traversal
3. In-order traversal
4. Post-order traversal
5. Cari node
6. Hapus subtree
7. Hitung kedalaman maksimal
8. Keluar
Pilih: 5
Masukkan label node yang dicari: g
Node dengan label g tidak ditemukan.
```

Kemudian ini tampilan saat kita mencari label yang kita ingin cari.

```
1. Tambah node
2. Pre-order traversal
3. In-order traversal
4. Post-order traversal
5. Cari node
6. Hapus subtree
7. Hitung kedalaman maksimal
8. Keluar
Pilih: 6
Masukkan label node yang akan dihapus subtree-nya: c
Subtree dengan root c telah dihapus.
```

```
1. Tambah node
2. Pre-order traversal
3. In-order traversal
4. Post-order traversal
5. Cari node
6. Hapus subtree
7. Hitung kedalaman maksimal
8. Keluar
Pilih: 6
Masukkan label node yang akan dihapus subtree-nya: g
Node tidak ditemukan!
```

Ini tampilan saat kita mengahpus subtreenya.

```
1. Tambah node
2. Pre-order traversal
3. In-order traversal
4. Post-order traversal
5. Cari node
6. Hapus subtree
7. Hitung kedalaman maksimal
8. Keluar
Pilih: 2
Pre-order traversal: a b d e
```

Dan ini setelah subtreenya terhapus.

```
1. Tambah node
2. Pre-order traversal
3. In-order traversal
4. Post-order traversal
5. Cari node
6. Hapus subtree
7. Hitung kedalaman maksimal
8. Keluar
Pilih: 7
Kedalaman maksimal tree: 3
```

Ini adalah tampilan menghitung kedalaman maksimal sebuah tree.

```
1. Tambah node
2. Pre-order traversal
3. In-order traversal
4. Post-order traversal
5. Cari node
6. Hapus subtree
7. Hitung kedalaman maksimal
8. Keluar
Pilih: 8
Program dibuat oleh : Zaldy Seno Yudhanto (2310631170123)

Process returned 0 (0x0)   execution time : 18.747 s
Press any key to continue.
```

Dan ini adalah tampilan saat kita keluar dari programnya.

Ini adalah link github tugas saya:

https://github.com/Dokii06/Semester-2/tree/main