**LAPORAN TUGAS STRUKTUR DATA**

**APLIKASI PENERAPAN *BINARY TREE***



Dosen Pembimbing:

Dr. Ricky Eka Putra, S.Kom., M.Kom.

Disusun oleh:

Cornelius Louis Nathan 23051204085

**PROGRAM STUDI S1 TEKNIK INFORMATIKA**

**UNIVERSITAS NEGERI SURABAYA**

**2024**

# KATA PENGANTAR

Puji syukur saya haturkan kepada Tuhan Yang Maha Esa sehingga atas rahmat-Nya. Saya dapat menyelesaikan laporan tugas struktur data yang berjudul “Aplikasi Penerapan *Binary Tree*” sebagai tugas struktur data tepat pada waktunya.

Adapun tujuan penulisan laporan ini adalah untuk mengetahui pengaplikasian *Tree* dan penerapan *binary tree* dalam suatu aplikasi. Pembuatan laporan ini sekaligus menjadi tugas yang mengisi nilai dari Mata Kuliah Struktur Data.

Penyusunan laporan ini tidak lepas dari bantuan dan dukungan. Sehingga, kami menyampaikan terima kasih kepada pihak yang telah mendukung pembuatan laporan ini. Terutama kepada Bapak Ricky Eka Putra, S.Kom., M.Kom., selaku Dosen Pembimbing Mata Kuliah Struktur Data, atas petunjuk, didikan, dan arahan yang sangat membantu dalam penyusunan laporan ini.

Kami menyadari bahwasanya penulisan laporan ini jauh dari kata sempurna, sehingga kami menerima saran dan kritik yang membangun dari pembaca guna untuk perbaikan dan perkembangan di masa mendatang. Semoga penyusunan laporan ini juga berguna bagi para pembaca dalam memahami pengaplikasian *tree* dan penerapan *binary tree* dalam suatu aplikasi.

Surabaya, 29 April 2024

Penulis

# DAFTAR ISI

[KATA PENGANTAR 2](#_Toc165713571)

[DAFTAR ISI 3](#_Toc165713572)

BAB I [PENDAHULUAN 4](#_Toc165713574)

[1.1 Latar Belakang 4](#_Toc165713575)

[1.2 Rumusan Masalah 4](#_Toc165713576)

[1.3 Tujuan 4](#_Toc165713577)

BAB II [KAJIAN PUSTAKA 5](#_Toc165713579)

[2.1 Memahami *Binary Tree* dan Komponen yang Dimiliki *Binary Tree* 5](#_Toc165713580)

[2.2 Pengimplementasian *Binary Tree* dalam kasus dunia nyata 6](#_Toc165713581)

[2.3 *Output* yang dihasilkan dari program tersebut 15](#_Toc165713582)

# BAB I

# PENDAHULUAN

## 1.1 Latar Belakang

Penerapan struktur data dalam dunia komputer dan pemrograman sangat diperlukan untuk memecahkan berbagai masalah kompleks dan mempermudah kinerja program atau sistem operasi untuk bekerja lebih efisien. Salah satu struktur data yang sering digunakan adalah *tree*. *tree* sendiri adalah struktur data yang terdiri atas akar (*root*) dan subpohon-subpohon dalam susunan berhierarki. *tree* sendiri memiliki banyak tipenya, yaitu *ternary tree, n-ary tree,* dan *binary tree. binary tree* memiliki ciri khas yang unik, dimana setiap simpul (*node*) memiliki paling banyak hanya dua anak, yaitu anak kiri dan anak kanan. Sehingga *binary Tree* terkesan lebih tertata apabila dibandingkan dengan *tree* yang lain.

Dalam kasus ini, andaikan seorang remaja penggemar *game arcade* bernama Julian mengoleksi banyak kartu dari permainan *game arcade* bernama RoboFight. Setiap kartu dari RoboFight memiliki kode unik, gambar robot, *power* dan *hitpoint (HP)* robot yang masing-masing berbeda satu dengan yang lain. Julian berhasil membuat aplikasi yang mampu untuk mengurutkan dan mencari kartu berdasarkan kode kartu, power, dan *HP* dari suatu robot. Teman Julian, Morris, ingin dibuatkan aplikasi yang sama seperti milik Julian. Oleh karena Moris adalah teman baik Julian, maka Julian bersedia untuk membuat aplikasi yang sama dengan penerapan *binary tree*.

## 1.2 Rumusan Masalah

1. Apa itu *binary tree* dan apa saja komponen yang dimiliki *binary tree*?
2. Bagaimana pengimplementasian *binary tree* dalam kasus dunia nyata?
3. Bagaimana *output* yang dihasilkan dari program tersebut?

## 1.3 Tujuan

1. Memahami *binary tree* dan komponen yang dimiliki *binary tree*.
2. Memahami pengimplementasian *binary tree* dalam kasus dunia nyata.
3. Memahami *output* yang dihasilkan dari program tersebut.

# BAB II

# KAJIAN PUSTAKA

## 2.1 Memahami *Binary Tree* dan Komponen yang Dimiliki *Binary Tree*

*Binary tree* adalah struktur data yang terdiri atas akar (*root*) dan subpohon (*subtree*) dalam susunan berhierarki. *Binary tree* memiliki ciri khas yang unik, dimana setiap simpul (*node*) memiliki paling banyak hanya dua anak (*child*), yaitu anak kiri dan anak kanan. Suatu *tree* yang kosong bisa dikatakan sebagai *binary tree. Binary tree* juga harus memenuhi salah satu dari syarat berikut :

* Tidak memiliki *child*,
* Memiliki *subtree* di sebelah kiri (*left subtree*)
* Memiliki *subtree* di sebelah kanan (*right subtree*)
* Memiliki *left subtree* maupun *right subtree*.

Suatu *tree* memiliki komponen yang saling berkaitan satu dengan yang lain, tak terkecuali dengan *binary tree*. *Parent* atau *internal node* adalah *node* yang digambarkan di atas *node* lain. *Root* utama tidak memiliki *parent. Child* adalah *node* yang digambarkan di bawah *node* lain. Suatu *node* bisa memiliki 0 sampai 2 *child*. *Child* yang memiliki nilai lebih kecil dari *parent* nya, akan terletak di sebelah kiri *parent*. Sedangkan *child* yang memiliki nilai lebih besar dari *parent* nya, akan terletak di sebelah kanan *parent*. Sehingga, apabila kita menginputkan suatu data, maka data tersebut otomatis akan terurut.

*Leaf* atau *external node* adalah *node* terujung yang tidak memiliki *child. Sibling* adalah *node* yang mempunyai *parent* yang sama. *Ancestor* atau leluhur adalah *node-node* terkait yang terletak di bawah *node utama*. Sedangkan *descendant* atau keturunan adalah *node-node* terkait yang terletak di bawah *node utama*.

*Binary Tree* juga memiliki tingkat atau *level*, tingkatan tersebut dimulai dari 0, 1, 2, sampai seterusnya. Kedalaman (*depth)* dari *binary tree* dihitung dari 1, 2, 3, sampai seterusnya. Tinggi dari *binary tree* bisa dilihat dari banyaknya tingkatan yang dimiliki oleh suatu *binary tree*. Komponen terakhir dari *binary tree* adalah derajat, derajat dari suatu *binary tree* dilihat dari banyaknya *child* yang dimiliki oleh suatu *node*. Untuk *binary tree,* *node* bisa memiliki 0 sampai 2 derajat.

## 2.2 Pengimplementasian *Binary Tree* dalam kasus dunia nyata

Andaikan seorang remaja penggemar *game arcade* bernama Julian mengoleksi banyak kartu dari permainan *game arcade* bernama RoboFight. Setiap kartu dari RoboFight memiliki kode unik, gambar robot, *power* dan *hitpoint (HP)* robot yang masing-masing berbeda satu dengan yang lain. Julian berhasil membuat aplikasi yang mampu untuk mengurutkan dan mencari kartu berdasarkan kode kartu, power, dan *HP* dari suatu robot.

Teman Julian, Morris, ingin dibuatkan aplikasi yang sama seperti milik Julian. Oleh karena Moris adalah teman baik Julian, maka Julian bersedia untuk membuat aplikasi yang sama dengan penerapan *binary tree*. Tetapi, Morris hanya ingin bisa menginputkan dan mengakses kartu berdasarkan kode kartu, tidak perlu menginputkan *power* dan *HP* dari robot tersebut.

Berikut adalah data kartu yang dimiliki oleh Morris :

134, 413, 351, 194, 256, 281, 180, 323, 418, 504, 274, 409, 341, 291, 315, 131, 473, 582, 541, 473, 261, 380, 174, 164

Dengan aplikasi yang dibuat oleh Julian, diharapkan Morris dapat

1. Menginputkan kode kartu dengan *binary tree*
2. Menampilkan kartu dengan tiga cara *traversal* 
   1. Pre Order
   2. In Order
   3. Post Order
3. Menghapus kode kartu dari *binary tree*

Tetapi, disini Julian menambahkan fitur tambahan untuk membantu Morris, yaitu :

1. Mencari kode kartu yang diinginkan
2. Menghitung berapa banyak kartu yang dimiliki
3. Menemukan kode kartu paling pertama dan menemukan kode kartu paling terakhir

Aplikasi dimulai dengan menginisiasi *struct tree* untuk menjadi *binary tree,* membuat fungsi *void create* untuk membuat *binary* tree, membuat fungsi *void clear* untuk menghapus semua data dalam *binary tree,* dan membuat fungsi *int IsEmpty* untuk memeriksa apakah *binary tree* kosong atau tidak, jika kosong akan mengembalikan nilai 1, jika *binary tree* memiliki data, akan mengembalikan nilai 0.

struct Tree{

    int Data;

    Tree \*Left;

    Tree \*Right;

};

Tree \*Root;

void Create(Tree \*Root){

    Root = nullptr;

}

void Clear(Tree \*Root){

    Root = nullptr;

}

int isEmpty(Tree \*Root){

    if(Root == nullptr) return 1;

    else return 0;

}

Membuat fungsi *void Insert* untuk menginputkan data.

void Insert(Tree \*\*Root, int NewData){

    if((\*Root) == nullptr){ //jika Root masih kosong

        Tree \*NewNode; // maka membuat Root baru pertama

        NewNode = new Tree;

        NewNode->Data = NewData;

        NewNode->Left = nullptr;

        NewNode->Right = nullptr;

        (\*Root) = NewNode;

        (\*Root)->Left = nullptr;

        (\*Root)->Right = nullptr;

    }

    // jika sebelumnya sudah ada data di dalam binary tree, dilakukan pengecekan

    else if(NewData < (\*Root)->Data){ // jika data yang baru lebih kecil dari Root

        Insert(&(\*Root)->Left, NewData); // maka mengakses subtree sebelah kiri

    }

    else if(NewData > (\*Root)->Data){ // jika data yang baru lebih besar dari Root

        Insert(&(\*Root)->Right, NewData);  // maka mengakses subtree sebelah kanan

    }

    else if(NewData == (\*Root)->Data){ // jika sebelumnya sudah menginputkan data yang sama

        cout << "Data has been inputted\n"; // maka data tidak bisa diinputkan

    }

}

Membuat fungsi *void PreOrder, InOrder, PostOrder* untuk menampilkan data kartu yang telah diinputkan oleh pengguna. Yang merupakan tiga proses *traverse.*

void PreOrder(Tree \*Root){

    if(Root == nullptr){

        return;

    }

    cout << Root->Data << " "; // menampilkan data Root

    PreOrder(Root->Left); // traverse ke kiri

    PreOrder(Root->Right); // traverse ke kanan

}

void InOrder(Tree \*Root){

    if(Root == nullptr){

        return;

    }

    InOrder(Root->Left); // traverse ke kiri

    cout << Root->Data << " "; // menampilkan data Root

    InOrder(Root->Right); // traverse ke kanan

} // sehingga menampilkan data secara terurut

void PostOrder(Tree \*Root){

    if(Root == nullptr){

        return;

    }

    PostOrder(Root->Left); // traverse ke kiri

    PostOrder(Root->Right);  // traverse ke kanan

    cout << Root->Data << " "; // menampilkan data Root

}

Menghapus kode kartu dari *binary tree.*

Tree \*DeleteNode(Tree \*Root, int DeleteData){ // mencari data yang akan dihapus

    if (Root == nullptr){ // jika Root kosong atau data yang dicari tidak ditemukan

        return nullptr; // maka akan mengembalikan nilai kosong

    }

    else if(DeleteData < Root->Data){ // jika data yang dicari lebih kecil dari Root

        Root->Left = DeleteNode(Root->Left, DeleteData); // maka mengakses subtree sebelah kiri

    }

    else if(DeleteData < Root->Data){  // jika data yang dicari lebih besar dari Root

        Root->Right = DeleteNode(Root->Right, DeleteData); // maka mengakses subtree sebelah kanan

    } // jika data memang sudah ditemukan

    else{ // jika Root memiliki 1 child saja

        if(Root->Left == nullptr){  // jika Root tidak memiliki child di sebelah kiri

            Tree \*Temp = Root->Right; // maka Temp akan menyimpan child sebelah kanan

            delete Root; // menghapus data

            return Temp; // maka Temp disambungkan kembali terhadap binary tree

        }

        if(Root->Right == nullptr){ // jika Root tidak memiliki child di sebelah kanan

            Tree \*Temp = Root->Left; // maka Temp akan menyimpan child sebelah kiri

            delete Root; // menghapus data

            return Temp; // maka Temp disambungkan kembali terhadap binary tree

        }

        else{ //jika Root memiliki 2 child

            Tree \*Temp = FindMax(Root->Left); // maka data nilai digantikan dengan nilai terbesar dari left subtree

            Root->Data = Temp->Data; // data disimpan di Temp

            Root->Left = DeleteNode(Root->Left,Temp->Data); // maka dilakukan penghapusan

        }

    }

    return Root;

}

Untuk tambahan fitur dari aplikasi tersebut, maka Julian juga membuat fungsi untuk mencari kode kartu yang diinginkan.

Tree \*Find(Tree \*Root, int FindData){

    if(Root == nullptr){ // jika Root kosong atau data tidak ditemukan

        return nullptr; // maka akan mengembalikan nilai kosong

    }

    else if(FindData < Root->Data) return Find(Root->Left,FindData); // jika data yang dicari lebih kecil dari Root, maka mencari ke child sebelah kiri

    else if(FindData > Root->Data) return Find(Root->Right,FindData); // jika data yang dicari lebih besar dari Root, maka mencari ke child sebelah kanan

    else return Root; // jika data sudah ditemukan, maka mengembalikan data yang dicari

}

Menghitung banyaknya kartu yang dimiliki oleh pengguna

int Count(Tree \*Root){

    if(Root == nullptr) return 0; //jika Root kosong, maka mengembalikan nilai nol (karena tidak ada kartu)

    return Count(Root->Left) + Count(Root->Right) + 1; // jika ada data, maka menghitung banyaknya node di left subtree dan right subtree, lalu ditambah 1 yang merupakan node root

}

Menemukan kode kartu paling pertama dan menemukan kode kartu paling terakhir

Tree \*FindMin(Tree \*Root){ // mencari data paling kecil

    if(Root == nullptr){ // jika Root kosong atau data tidak ditemukan

        return nullptr; // maka akan mengembalikan nilai kosong

    }

    else{

        if(Root->Left == nullptr) return Root; // jika subtree left kosong, berarti data tersebutlah yang paling kiri (yang paling kecil)

        else return FindMin(Root->Left); // jika memang belum kosong, maka mengakses subtree sebelah kiri

    }

}

Tree \*FindMax(Tree \*Root){ // mencari data paling kecil

    if(Root == nullptr){ // jika Root kosong atau data tidak ditemukan

        return nullptr; // maka akan mengembalikan nilai kosong

    }

    else{

        if(Root->Right == nullptr) return Root; // jika subtree right kosong, berarti data tersebutlah yang paling kanan (yang paling besar)

        else return FindMax(Root->Right);  // jika memang belum kosong, maka mengakses subtree sebelah kanan

    }

}

Sebenarnya terdapat juga fungsi *int Height* untuk menghitung banyaknya tingkatan dalam *binary tree* dan fungsi *void FindLeaf* untuk mencari *leaf* paling ujung dari suatu *binary tree*.

int Height(Tree \*Root){

    if(Root == nullptr) return -1; // jika Root kosong, maka mengembalikan nilai -1 (karena tidak ada tinggi)

    int LeftHeight = Height(Root->Left); // dilakukan pembagian LeftHeight untuk menghitung kedalaman dari subtree sebelah kiri

    int RightHeight = Height(Root->Right); // dilakukan pembagian RightHeight untuk menghitung kedalaman dari subtree sebelah kanan

    if(LeftHeight > RightHeight) return LeftHeight + 1; // jika LeftHeight lebih besar, maka digunakan penghitungan LeftHeight + 1 (yg merupakan root utama)

    else return RightHeight + 1; // jika RightHeight lebih besar, maka digunakan penghitungan RightHeight + 1 (yg merupakan root utama)

}

void FindLeaf(Tree \*Root){

    if(Root == nullptr){

        return;

    }

    if(Root->Left != nullptr) FindLeaf(Root->Left); // jika masih terdapat child di sebelah kiri, maka mengakses subtree sebelah kiri

    if(Root->Right != nullptr) FindLeaf(Root->Right); // jika masih terdapat child di sebelah kanan, maka mengakses subtree sebelah kanan

    else if(Root->Left == nullptr && Root->Right == nullptr){ // jika sudah tidak ada child di sebelah kiri atau kanan

        cout << "The leaf is " << Root->Data << endl; //maka leaf ditemukan

    }

}

Setelah membuat semua fungsi tersebut, maka waktunya membuat program utama dari aplikasi RoboFight, sehingga pengguna dapat mengakses semua fungsi-fungsi yang telah dibuat sebelumnya.

int main(){

    Create(Root);

    int Choice;

    char Login = 'y';

    while(Login == 'y'){

        system("cls");

        cout << "========== RoboFight App ==========\n\n";

        cout << "1. Enter your RoboFight card code\n";

        cout << "2. See all your RoboFight cards collection\n";

        cout << "3. Delete your RoboFight card code from your collection\n";

        cout << "4. Find your RoboFight card code\n";

        cout << "5. How many cards do you have?\n";

        cout << "6. What is your first and last card code card?\n";

        cout << "7. Exit\n";

        cout << "> ";

        cin >> Choice;

Pada menu *1. Enter your RoboFight Card,* pengguna bisa menginputkan data kartu-kartu yang dimiliki berdasarkan kode kartu. Data kartu yang telah diinputkan akan masuk ke dalam *binary tree*.

switch (Choice){

        case 1:{

            int CardCode;

            cout << "Enter your RoboFight card code > ";

            cin >> CardCode;

            Insert(&Root,CardCode);

            cin.ignore();

            cout << "\nYour card code has been saved\n";

            cout << "Press Enter to Continue\n";

            cin.get();

        }

        break;

Pada menu *2. See All your RoboFight Cards Collection*, pengguna dapat memilih 3 cara untuk menampilkan semua kartu yang dimiliki, yaitu secara *Ppe order, in order,* dan *post order*. Setelah menginputkan metode yang diinginkan, sesuai dengan nomornya, program akan menampilkan semua kode kartu yang dimiliki sesuai dengan algoritma dari masing-masing fungsi.

case 2:{

            int TraversalChoice;

            cout << "Select the method for seeing all your cards\n";

            cout << "1. Pre Order\n";

            cout << "2. In Order\n";

            cout << "3. Post Order\n";

            cout << "> ";

            cin >> TraversalChoice;

cout << "\nList of your collection : \n";

if(TraversalChoice == 1){

                PreOrder(Root);

            }

            else if(TraversalChoice == 2){

                InOrder(Root);

            }

            else if(TraversalChoice == 3){

                PostOrder(Root);

            }

            cin.ignore();

            cout << "\n\nPress Enter to Continue\n";

            cin.get();

        }

        break;

Pada menu 3. *Delete your RoboFight card code from your collection*¸ pengguna dapat memasukkan kode kartu yang ingin dihapus. Bila kode kartu yang ingin dihapus telah ditemukan, maka penghapusan dapat dilakukan. Namun, apabila kode kartu yang ingin dihapus tidak ditemukan, maka proses penghapusan akan gagal.

case 3:{

            int DeleteCode;

            cout << "Enter the card code you wish to delete > ";

            cin >> DeleteCode;

            Root = DeleteNode(Root, DeleteCode);

            cout << "\nCard code with code " << DeleteCode << " has been deleted from your collection\n \n";

            cout << "List of your collection : \n";

            InOrder(Root);

            cin.ignore();

            cout << "\n\nPress Enter to Continue\n";

            cin.get();

        }

        break;

Pada menu 4. *Find your RoboFight card code,* penggunadapat menginputkan kode kartu yang ingin dicari. Jika kode kartu telah ditemukan, maka program akan memberitahu bahwa pengguna memiliki kode kartu tersebut. Tapi bila tidak ditemukan, maka program akan memberitahu bahwa pengguna tidak memiliki kode kartu tersebut.

case 4:{

            int FindCode;

            cout << "Enter your RoboFight card code > ";

            cin >> FindCode;

            Tree \*Result = Find(Root, FindCode);

            if(Result != nullptr){

                cout << "You have card with code " << FindCode << endl;

            }

            else{

                cout << "You do not have card with code " << FindCode << endl;

            }

            cin.ignore();

            cout << "\nPress Enter to Continue\n";

            cin.get();

        }

        break;

Pada menu 5. *How many cards do you have?,* fungsi *int Count* akan melakukan penghitungan berapa banyak kode kartu yang telah diiinputkan oleh pengguna.

case 5:{

            int CardCount = Count(Root);

            cout << "You have " << CardCount << " cards in your collection\n";

            cin.ignore();

            cout << "\nPress Enter to Continue\n";

            cin.get();

        }

        break;

Pada menu 6.  *What is your first card code and your last card code?*, fungsi *tree \*FindMin* dan *tree \*FindMax* akan melakukan pencarian data terkecil dan terbesar.

case 6:{

            Tree \*FirstCardCode = FindMin(Root);

            Tree \*LastCardCode = FindMax(Root);

            cout << "Your first card code is " << FirstCardCode->Data << endl;

            cout << "Your last card code is " << LastCardCode->Data << endl;

            cin.ignore();

            cout << "\nPress Enter to Continue\n";

            cin.get();

        }

        break;

Yang terakhir, pada menu *7. Exit*, pengguna dapat memilih untuk keluar dari aplikasi. Apabila pengguna tetap ingin keluar, maka perlu mengetikkan huruf ‘y’ atau ‘Y’. Namun bila tidak ingin keluar, maka pengguna dapat mengetikkan huruf selain ‘Y’ atau ‘y’.

case 7:{

            char Exit;

            cout << "Are you sure you to exit the application? (y/n)\n";

            cout << "> ";

            cin >> Exit;

            if(Exit == 'y' || Exit == 'Y'){

                Login = 'n';

            }

            else{

                Login = 'y';

            }

        }

        break;

Setiap fungsi diletakkan di akhir program sehingga, terdapat *prototype fungsi* pada awal program.

struct Tree{

    int Data;

    Tree \*Left;

    Tree \*Right;

};

Tree \*Root;

void Create(Tree \*Root);

void Clear(Tree \*Root);

void IsEmpty(Tree \*Root);

void Insert(Tree \*\*Root, int NewData);

void PreOrder(Tree \*Root);

void InOrder(Tree \*Root);

void PostOrder(Tree \*Root);

Tree \*DeleteNode(Tree \*Root, int DeleteData);

Tree \*Find(Tree \*Root, int FindData);

int Count(Tree \*Root);

Tree \*FindMin(Tree \*Root);

Tree \*FindMax(Tree \*Root);

int Height(Tree \*Root);

void FindLeaf(Tree\*Root);

## 2.3 *Output* yang dihasilkan dari program tersebut

Setelah menjalankan dan menginputkan data sesuai dengan tabel sebelumnya, program dapat berjalan dengan baik dan dapat memberikan output yang diharapkan. Berikut adalah hasil input kode kartu berdasarkan table sebelumnya.

========== RoboFight App ==========

1. Enter your RoboFight card code

2. See all your RoboFight cards collection

3. Delete your RoboFight card code from your collection

4. Find your RoboFight card code

5. How many cards do you have?

6. What is your first and last card code card?

7. Exit

> 1

Enter your RoboFight card code > 134

Your card code has been saved

Press Enter to Continue

Maka dilakukan penginputan kode kartu secara terus menerus sampai semua kode kartu telah berhasil diinputkan.

Untuk menampilkan semua kode kartu yang dimiliki dengan proses *traverse*, yaitu *pre order, in order, dan post order*. Berikut adalah output yang dihasilkan dengan cara *pre order*:

========== RoboFight App ==========

1. Enter your RoboFight card code

2. See all your RoboFight cards collection

3. Delete your RoboFight card code from your collection

4. Find your RoboFight card code

5. How many cards do you have?

6. What is your first and last card code card?

7. Exit

> 2

Select the method for seeing all your cards

1. Pre Order

2. In Order

3. Post Order

> 1

List of your collection:

134 131 413 351 194 180 174 164 256 281 274 261 323 291 315 341 409 380 418 504 473 582 541

Press Enter to Continue

Berikut adalah output yang dihasilkan dengan cara *in order*:

========== RoboFight App ==========

1. Enter your RoboFight card code

2. See all your RoboFight cards collection

3. Delete your RoboFight card code from your collection

4. Find your RoboFight card code

5. How many cards do you have?

6. What is your first and last card code card?

7. Exit

> 2

Select the method for seeing all your cards

1. Pre Order

2. In Order

3. Post Order

> 2

List of your collection:

131 134 164 174 180 194 256 261 274 281 291 315 323 341 351 380 409 413 418 473 504 541 582

Press Enter to Continue

Berikut adalah output yang dihasilkan dengan cara *post order*:

========== RoboFight App ==========

1. Enter your RoboFight card code

2. See all your RoboFight cards collection

3. Delete your RoboFight card code from your collection

4. Find your RoboFight card code

5. How many cards do you have?

6. What is your first and last card code card?

7. Exit

> 2

Select the method for seeing all your cards

1. Pre Order

2. In Order

3. Post Order

> 3

List of your collection:

131 164 174 180 261 274 315 291 341 323 281 256 194 380 409 351 473 541 582 504 418 413 134

Press Enter to Continue

Untuk menghapus suatu kode kartu dari koleksi kartu tersebut, program juga dapat menghapus dengan baik. Andaikan pengguna ingin menghapus kode kartu 261, maka angka 261 akan dihapus dari koleksi. Maka berikut adalah output yang dihasilkan :

========== RoboFight App ==========

1. Enter your RoboFight card code

2. See all your RoboFight cards collection

3. Delete your RoboFight card code from your collection

4. Find your RoboFight card code

5. How many cards do you have?

6. What is your first and last card code card?

7. Exit

> 3

Enter the card code you wish to delete > 261

Card code with code 261 has been deleted from your collection

List of your collection :

131 164 174 180 194 256 274 281 291 315 323 341 351 380 409 413 418 473 504 541 582

Press Enter to Continue

Jika pengguna ingin mencari kode kartu yang diinginkan, maka pengguna dapat mengakses menu 4 dan program akan memberitahu bahwa pengguna memiliki kode kartu tersebut atau tidak. Berikut adalah output yang dihasilkan :

========== RoboFight App ==========

1. Enter your RoboFight card code

2. See all your RoboFight cards collection

3. Delete your RoboFight card code from your collection

4. Find your RoboFight card code

5. How many cards do you have?

6. What is your first and last card code card?

7. Exit

> 4

Enter your RoboFight card code > 274

You have card with code 274

Press Enter to Continue

Namun, apabila pengguna tidak memiliki kode kartu yang dicari, maka berikut adalah output yang dihasilkan :

========== RoboFight App ==========

1. Enter your RoboFight card code

2. See all your RoboFight cards collection

3. Delete your RoboFight card code from your collection

4. Find your RoboFight card code

5. How many cards do you have?

6. What is your first and last card code card?

7. Exit

> 4

Enter your RoboFight card code > 500

You do not have card with code 500

Press Enter to Continue

Jika pengguna ingin mengetahui berapa banyak kartu yang dimiliki oleh pengguna, Maka program akan melakukan penghitungan dan menampilkan banyaknya kartu yang dimiliki. Berikut adalah output yang dihasilkan :

========== RoboFight App ==========

1. Enter your RoboFight card code

2. See all your RoboFight cards collection

3. Delete your RoboFight card code from your collection

4. Find your RoboFight card code

5. How many cards do you have?

6. What is your first and last card code card?

7. Exit

> 5

You have 22 cards in your collection

Press Enter to Continue

Jika pengguna ingin mengetahui kode kartu pertama dan kode kartu terakhir yang dimiiki, maka pengguna dapat mengakses menu nomor 6. Berikut adalah output yang dihasilkan :

========== RoboFight App ==========

1. Enter your RoboFight card code

2. See all your RoboFight cards collection

3. Delete your RoboFight card code from your collection

4. Find your RoboFight card code

5. How many cards do you have?

6. What is your first and last card code card?

7. Exit

> 6

Your first card code is 131

Your last card code is 582

Press Enter to Continue

Jika pengguna ingin keluar dari aplikasi, maka cukup mengakses menu nomor 7. Berikut adalah output yang dihasilkan :

========== RoboFight App ==========

1. Enter your RoboFight card code

2. See all your RoboFight cards collection

3. Delete your RoboFight card code from your collection

4. Find your RoboFight card code

5. How many cards do you have?

6. What is your first and last card code card?

7. Exit

> 7

Are you sure you to exit the application? (y/n)

> y