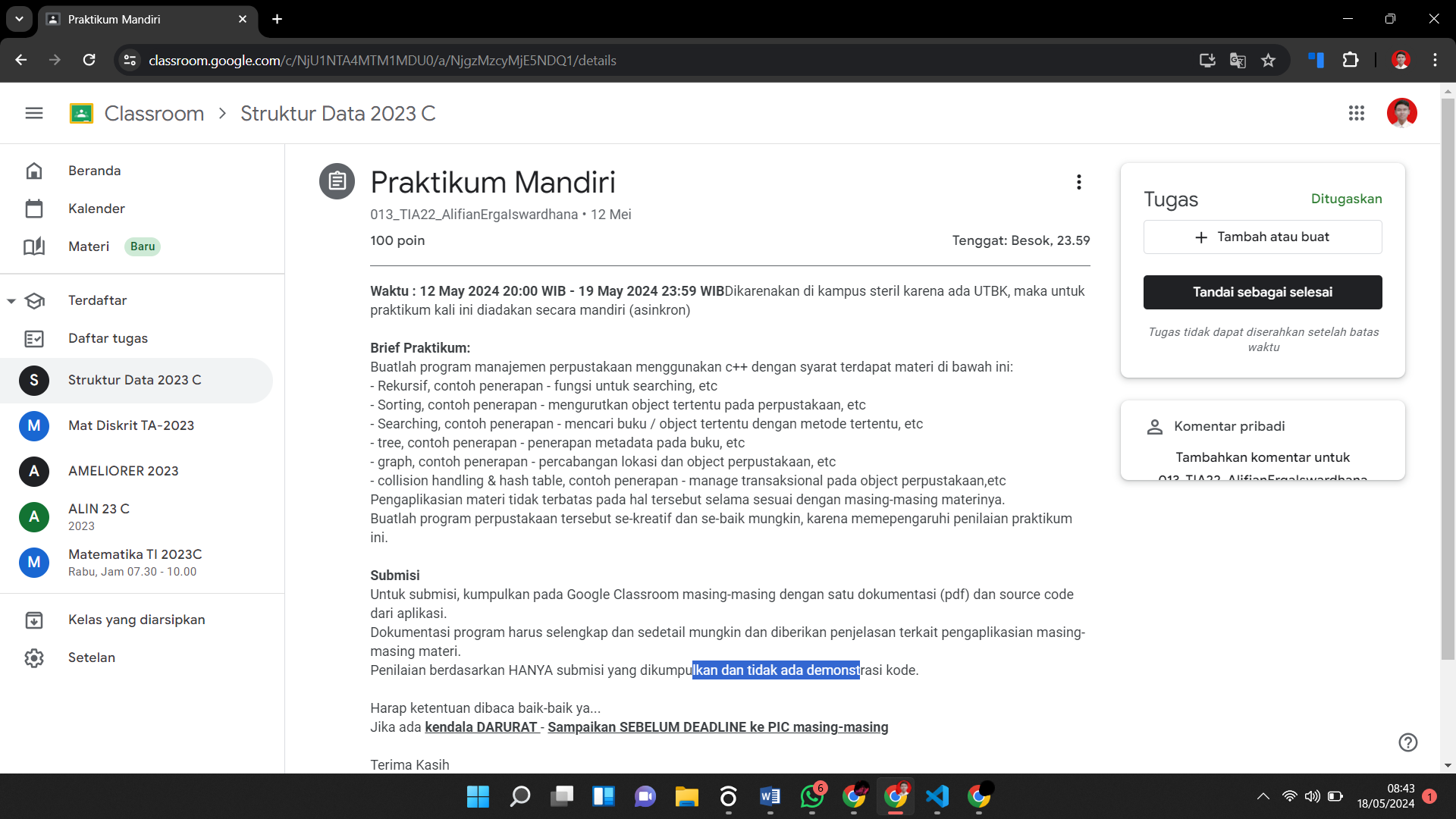
Dokumentasi Program 085\_Cornelius Louis Nathan



Untuk mendukung pengaplikasikan 6 materi tersebut, maka program saya menggunakkan 7 *library*, berikut adalah penjelasan dari *library* tersebut:

#include <iostream> // library untuk menampilkan dan menginput

#include <vector> // library dynamic array

#include <string> // library untuk pengaturan string (membantu proses pembuatan kode buku)

#include <unordered\_map> // library untuk menghubungkan vertex satu dengan vertex lainnya (membantu proses pembuatan GRAPH)

#include <stack> // library stack (membantu proses pembuatan GRAPH DFS)

#include <list> // library pembuatan tabel hash (membantu proses TABLE HASH (COLLISION HANDLING))

#include <ctime> // library pengaksesan waktu (membantu proses pembuatan string tanggal peminjaman buku pada TABLE HASH (COLLISION HANDLING))

using namespace std;

Program saya juga memiliki 3 buah struktur data yaitu *Vector, Tree, dan Table Hashing*.

* *Vector (Struct Dynamic Array)* digunakan untuk menyimpan data buku dan mengurutkan data buku berdasarkan judul,
* *Tree* digunakan untuk menyimpan data buku dan mengurutkan data buku berdasarkan author (penulis),
* *Table Hashing* digunakan untuk menyimpan data peminjaman buku.

*Vector* memiliki anggota *struct* sebagai berikut:

struct BookData{ // struct vector (dynamic array) untuk menyimpan data buku

    string CodeBook; // kode buku di perpus

    string Title; // nama judul

    string Author; // nama penulis

    int Year; // tahun perilisan buku

};

*Tree* memiliki anggota *struct* sebagai berikut:

struct TreeBook{ // tree untuk penyimpanan data buku (untuk pencarian author)

    BookData Data; // data buku

    TreeBook \*Left; // child sebelah kiri (memiliki urutan huruf lebih kecil)

    TreeBook \*Right; // child sebelah kanan (memiliki urutan huruf lebih kecil)

};

*Table Hashing* memiliki anggota *struct* sebagai berikut:

struct BorrowerData{ // data peminjam yang akan dimasukkan table hash

    BookData Book; // data buku

    string BorrowerName; // nama peminjam

    string BorrowDate; // tanggal peminjaman buku

};

Andaikan terdapat data buku sebagai berikut:

{"022014", "Bumi", "Tere Liye", 2014},

{"003188", "Clean Code", "Robert Cecil Martin", 2008},

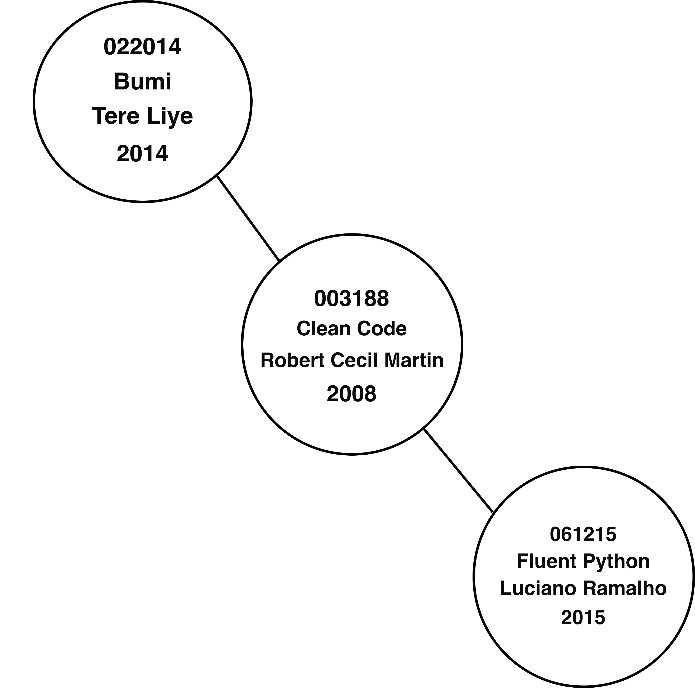
{"061215", "Fluent Python", "Luciano Ramalho", 2015},

Berikut adalah ilustrasi dari 3 buah struktur data tersebut

1. *Vector*



1. *Tree*



1. *Table Hash*

*Table Hash* digunakan untuk menyimpan data peminjaman buku

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No | Kode Buku | Judul Buku | Nama Peminjam | Tanggal Peminjaman |
| … |  |  |  |  |
| 13 | 003188 | Clean Code | Louis | 15-05-2024 |
| 14 | 022014 | Bumi | Farhan | 12-05-2024 |
| 15 | 061215 | Fluent Python | Adrian | 08-05-2024 |
| … |  |  |  |  |

\*003188 % 25 = 13

\*\*022014 % 25 = 14

\*\*\*061215 % 25 = 15

Program saya juga memiliki 26 fungsi untuk proses pengaplikasian 6 materi di semester 2.

void InputBookData(BookData& Book); // fungsi memasukkan data buku baru

string GenerateCodeBook(string Title,string Author,int Year); // fungsi membuat kode buku di perpus

void DeleteBookData(vector<BookData>& Book, string &TargetedTitle); // fungsi menghapus data buku

// FUNGSI TREE

void Create(TreeBook \*&Author); // fungsi membuat tree

int IsEmpty(TreeBook \*Author); // fungsi untuk memeriksa apakah tree kosong atau tidak

TreeBook\* InsertNode(TreeBook \*&Author, BookData &Book); // fungsi untuk menginputkan node ke dalam tree

void DisplayInOrder(TreeBook \*Author); // fungsi untuk menampilkan daftar buku secara urut berdasarkan nama depan author

TreeBook\* SearchAuthor(TreeBook \*Author, string &TargetedAuthor); // fungsi untuk mencari author buku menggunakkan BST (Binary Search Tree)

TreeBook\* DeleteNode(TreeBook \*Author,string &TargetedTitle); // fungsi untuk menghapus buku dari tree

TreeBook \*FindMax(TreeBook \*Author); // fungsi untuk mencari node paling ujung kanan (yang akan membantu fungsi DeleteNode)

// FUNGSI GRAPH

void AddLine(Graph &Recommendation, string From, string To);

void DFS(const Graph &Recommendation, const string &From);

// FUNGSI SORT

void SortBooksByTitle(vector<BookData>& CorneliusLibrary); // fungsi untuk mengurutkan buku berdasarkan judul

void QuickSort(vector<BookData>& CorneliusLibrary,int Low, int High); // menggunakkan fungsi quick sort

int Part(vector<BookData>& CorneliusLibrary,int Low, int High); // partisi pada quick sort

void Swap(BookData& i, BookData& j); // pertukaran posisi struct pada quick sort

// FUNGSI SEARCH

int BinarySearchByTitle(vector<BookData>& CorneliusLibrary, string &TargetedTitle);// fungsi untuk mencari judul menggunakkan binary search

string GetCurrentDate(); // fungsi untuk mendapatkan tanggal hari ini

// FUNGSI TABLE HASH DAN COLLISION HANDLING

int HashFunction(string &CodeBook); // fungsi hashing

void AddBook(BookData &Book); // fungsi memasukkan data buku ke dalam table hash

BookData\* FindBookBasedOnTitle(string &TargetedTitle); // fungsi mencari buku berdasarkan judul

void BorrowBookBasedOnTitle(string &TargetedTitle, string &BorrowerName, string &BorrowDate); // fungsi meminjam buku berdasarkan judul

BookData\* FindBookBasedOnAuthor(string &TargetedAuthor); // fungsi mencari buku berdasarkan author

void BorrowBookBasedOnAuthor(string &TargetedAuthor, string &BorrowerName, string &BorrowDate); // fungsi meminjam buku berdasarkan author

void DisplayBorrowList(); // fungsi untuk menampilkan semua data peminjaman

26 fungsi tersebut saya letakkan paling belakang setelah program utama agar program terlihat lebih rapi. Maka terdapat 26 fungsi *prototype* pada awal program.

Program saya juga memiliki 19 buku yang berisikan buku-buku serial milik Tere Liye dan beberapa buku pemrograman. Berikut adalah daftarnya

vector<BookData> CorneliusLibrary = {

    {"022014", "Bumi", "Tere Liye", 2014},

    {"022015", "Bulan", "Tere Liye", 2015},

    {"132016", "Matahari", "Tere Liye", 2016},

    {"022017", "Bintang", "Tere Liye", 2017},

    {"032018", "Ceros dan Batazoar", "Tere Liye", 2018},

    {"112018", "Komet", "Tere Liye", 2018},

    {"112019", "Komet Minor", "Tere Liye", 2019},

    {"192022", "Selena", "Tere Liye", 2020},

    {"142021", "Nebula", "Tere Liye", 2021},

    {"192022", "Si Putih", "Tere Liye", 2022},

    {"003188", "Clean Code", "Robert Cecil Martin", 2008},

    {"020199", "The Pragmatic Programmer", "Andy Hunt & Dave Thomas", 1999},

    {"201918", "The Rust Programming Language", "Steve Klabnik", 2018},

    {"111018", "Kotlin Programming: From Idea to Production", "JetBrains", 2018},

    {"003815", "C++: The Complete Reference", "Herbert Schildt", 2015},

    {"000339", "Competitive Programming: Algorithms and Techniques", "Cormen, Leiserson, Rivest, Stein", 2009},

    {"016515", "Python Crash Course", "Eric Matthes", 2015},

    {"001115", "Automate the Boring Stuff with Python", "Al Sweigart", 2015},

    {"061215", "Fluent Python", "Luciano Ramalho", 2015},

};

Pengaplikasian 6 materi semester 2 :

1. Rekursif

Rekursif adalah fungsi yang memanggil dirinya sendiri. Pada program saya, terdapat 6 fungsi yang merupakan fungsi rekursif.

1. Fungsi *Treebook\* InsertNode* untuk menginputkan *node* ke dalam *tree* menggunakkan konsep rekursif. Fungsi tersebut termasuk *direct non tail recursion.*

TreeBook\* InsertNode(TreeBook \*&Author,  BookData &Book){  // fungsi untuk menginputkan node ke dalam tree

    if(Author == nullptr){ //jika tree masih kosong

        TreeBook \*NewNode = new TreeBook; // maka membuat node pertama

        NewNode->Data = Book;

        NewNode->Left = nullptr;

        NewNode->Right = nullptr;

        Author = NewNode;

    }

    else if (Book.Title < Author->Data.Title){ // jika urutan huruf depan lebih kecil dari buku saat ini, maka melakukan pencarian ke kiri

        Author->Left = InsertNode(Author->Left, Book);

    }

    else if (Book.Title > Author->Data.Title){ // jika urutan huruf depan lebih besar dari buku saat ini, maka melakukan pencarian ke kanan

        Author->Right = InsertNode(Author->Right, Book);

    }

    return Author;

}

1. Fungsi *void DisplayInOrder* untuk menampilkan daftar buku secara urut berdasarkan menggunakkan konsep rekursif. Fungsi tersebut termasuk *direct tail recursion.*

void DisplayInOrder(TreeBook \*Author){ // fungsi untuk menampilkan daftar buku secara urut berdasarkan huruf depan buku

    if(Author == nullptr) return; // jika tree masih kosong, maka mengembalikan nilai kosong

    DisplayInOrder(Author->Left); // melakukan penelusuran ke kiri

    cout << Author->Data.Title << " written by " << Author->Data.Author << endl; // menampilkan data buku

    cout << "released in " << Author->Data.Year << " with code book " << Author->Data.CodeBook << endl << endl;

    DisplayInOrder(Author->Right); // melakukan penelusuran ke kanan

}

1. Fungsi *TreeBook\* SearchAuthor* untuk mencari *author* buku/penulis menggunakkan *BST (Binary Search Tree)* menggunakkan konsep rekursif. Fungsi tersebut termasuk *direct non tail recursion.*

TreeBook\* SearchAuthor(TreeBook \*Author, string &TargetedAuthor){ // fungsi untuk mencari author buku menggunakkan BST (Binary Search Tree)

    if(Author == nullptr) return nullptr;

    else if(TargetedAuthor < Author->Data.Author){ // jika urutan huruf depan lebih kecil dari tree, maka melakukan pencarian ke kiri

        SearchAuthor(Author->Left,TargetedAuthor);

    }

    else if(TargetedAuthor > Author->Data.Author){

        SearchAuthor(Author->Right,TargetedAuthor); // jika urutan huruf depan lebih kecil dari tree, maka melakukan pencarian ke kiri

    }

    return Author; // jika ditemukan, maka langsung mengembalikan data buku

}

1. Fungsi *TreeBook\* DeleteNode* untuk menghapus data buku dari dalam tree menggunakkan konsep rekursif. Fungsi tersebut termasuk *direct non tail recursion.*

TreeBook\* DeleteNode(TreeBook \*Author,string &TargetedTitle){ // fungsi untuk menghapus data buku

    if(Author == nullptr) return nullptr; // jika tree kosong, maka mengembalikan nilai kosong

    else if(TargetedTitle < Author->Data.Title){ // jika data yang dicari urutan hurufnya lebih kecil dari tree

        Author->Left = DeleteNode(Author->Left,TargetedTitle); // maka bergeser ke sebelah kiri

    }

    else if(TargetedTitle > Author->Data.Title){ // jika data yang dicari urutan hurufnya lebih kecil dari tree

        Author->Right = DeleteNode(Author->Right,TargetedTitle); // maka bergeser ke sebelah kanan

    }

    // jika data memang sudah ditemukan

    else{ // jika tree memiliki 1 child saja

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

            TreeBook \*Temp = Author->Right; // maka Temp akan menyimpan child sebelah kanan

            delete Author; // menghapus data

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

        }

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

            TreeBook \*Temp = Author->Left; // maka Temp akan menyimpan child sebelah kiri

            delete Author; // menghapus data

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

        }

        else{ //jika tree memiliki 2 child

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

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

            Author->Left = DeleteNode(Author->Left,Temp->Data.Title); // maka dilakukan penghapusan

        }

    }

    return Author; // mengembalikan tree

}

1. Fungsi *TreeBook\* FindMax* untuk mencari node paling ujung kanan menggunakkan konsep rekursif. Fungsi tersebut termasuk *direct tail recursion.*

TreeBook \*FindMax(TreeBook \*Author){ // fungsi untuk mencari node paling ujung kanan (yang akan membantu fungsi DeleteNode)

    if(Author == nullptr) return nullptr; // jika tree kosong, maka mengembalikan nilai kosong

    else{

        if(Author->Right == nullptr) return Author; // Jika sudah tidak terdapat child di sebelah kanan, maka data paling ujung kanan telah ditemukan

        else return FindMax(Author->Right); // jika belum, maka pelakukan penelusuran ke sebelah kanan

    }

}

1. Fungsi *void QuickSort* untuk melakukan pengurutan *quick sort*.

void QuickSort(vector<BookData>& CorneliusLibrary,int Low, int High){ // menggunakkan fungsi quick sort

    if(Low < High){

        int Pivot = Part(CorneliusLibrary,Low,High); // partisi pada quick sort

        QuickSort(CorneliusLibrary,Low,Pivot - 1); // rekursif quick sort

        QuickSort(CorneliusLibrary,Pivot + 1,High); // rekursif quick sort

    }

}

1. *Sorting*

*Sorting* adalah fungsi yang digunakan untuk melakukan pengurutan data berdasarkan kriteria tertentu. Pada program saya, fungsi *sorting* digunakan untuk mengurutkan judul buku. Fungsi *sorting* yang saya gunakan adalah *quick sort*.

*Quick Sort* adalah algoritma pengurutan yang menggunakkan pivot sebagai patokan. Data-data di dalam *dynamic array* akan dibandingkan dengan pivot, sehingga didapatkan data yang nilainya kurang dari pivot dan data yang nilainya lebih dari pivot. Pada kasus ini, data yang nilainya kurang adalah urutan hurufnya lebih kecil dari pivot. Lalu, data yang nilainya lebih dari pivot adalah urutan hurufnya lebih besar dari pivot.

Setelah itu, terjadi partisi yang dimana data yang nilainya kurang dari pivot terletak di sebelah kiri pivot, dan data yang nilainya lebih dari pivot terletak di sebelah kanan pivot. *Quick sort* yang saya gunakan terdiri atas 4 fungsi, keempat fungsi tersebut saling melengkapi satu dengan yang lain.

Awalnya saya menggunakkan fungsi *quick sort* sesuai dengan yang ada di dalam *powerpoint* yang diberikan oleh Pak Ricky. Tapi karena terdapat *bug* (dimana data tidak terurut dengan baik) dan ketidaksesuaian tipe data, maka saya ubah sesuai dengan kebutuhan tipe data.

// PENGAPLIKASIAN SORT

void SortBooksByTitle(vector<BookData>& CorneliusLibrary){ // fungsi untuk mengurutkan buku berdasarkan judul

    int Total = CorneliusLibrary.size(); // menghitung banyaknya buku yang ada dalam perpus

    QuickSort(CorneliusLibrary,0,Total-1);

}

void QuickSort(vector<BookData>& CorneliusLibrary,int Low, int High){ // menggunakkan fungsi quick sort

    if(Low < High){

        int Pivot = Part(CorneliusLibrary,Low,High); // partisi pada quick sort

        QuickSort(CorneliusLibrary,Low,Pivot - 1); // rekursif quick sort

        QuickSort(CorneliusLibrary,Pivot + 1,High); // rekursif quick sort

    }

}

int Part(vector<BookData>& CorneliusLibrary,int Low, int High){ // partisi pada quick sort

    string Pivot = CorneliusLibrary[High].Title; // buku paling terakhir pada perpus dijadikan pivot

    int i = Low - 1;

    for(int j = Low; j < High; j++){

        if(CorneliusLibrary[j].Title < Pivot){ // jika terdapat judul yang secara urutan huruf lebih kecil dari Pivot

            i++;

            Swap(CorneliusLibrary[i], CorneliusLibrary[j]); // maka ditukar

        }

    }

    Swap(CorneliusLibrary[i+1], CorneliusLibrary[High]); // pertukaran pivot ke posisi yang benar

    return i + 1;

}

void Swap(BookData& i, BookData& j){ // pertukaran posisi struct pada quick sort

    BookData Temp = i;

    i = j;

    j = Temp;

}

Berikut adalah hasil output dari *sorting* berdasarkan data buku di atas:

Book 1

    | Automate the Boring Stuff with Python | written by | Al Sweigart |

    released in 2015 with code book 001115

Book 2

    | Bintang | written by | Tere Liye |

    released in 2017 with code book 022017

Book 3

    | Bulan | written by | Tere Liye |

    released in 2015 with code book 022015

Book 4

    | Bumi | written by | Tere Liye |

    released in 2014 with code book 022014

Book 5

    | C++: The Complete Reference | written by | Herbert Schildt |

    released in 2015 with code book 003815

Book 6

    | Ceros dan Batazoar | written by | Tere Liye |

    released in 2018 with code book 032018

Book 7

    | Clean Code | written by | Robert Cecil Martin |

    released in 2008 with code book 003188

Book 8

    | Competitive Programming: Algorithms and Techniques | written by | Cormen, Leiserson, Rivest, Stein |

    released in 2009 with code book 000339

Book 9

    | Fluent Python | written by | Luciano Ramalho |

    released in 2015 with code book 061215

Book 10

    | Komet | written by | Tere Liye |

    released in 2018 with code book 112018

Book 11

    | Komet Minor | written by | Tere Liye |

    released in 2019 with code book 112019

Book 12

    | Kotlin Programming: From Idea to Production | written by | JetBrains |

    released in 2018 with code book 111018

Book 13

    | Matahari | written by | Tere Liye |

    released in 2016 with code book 132016

Book 14

    | Nebula | written by | Tere Liye |

    released in 2021 with code book 142021

Book 15

    | Python Crash Course | written by | Eric Matthes |

    released in 2015 with code book 016515

Book 16

    | Selena | written by | Tere Liye |

    released in 2020 with code book 192020

Book 17

    | Si Putih | written by | Tere Liye |

    released in 2022 with code book 192022

Book 18

    | The Pragmatic Programmer | written by | Andy Hunt & Dave Thomas |

    released in 1999 with code book 020199

Book 19

    | The Rust Programming Language | written by | Steve Klabnik |

    released in 2018 with code book 201918

Press Enter to continue

1. *Searching*

*Searching* adalah fungsi yang digunakan untuk melakukan proses pencarian data yang diinginkan. Pada program saya, fungsi *searching* digunakan untuk mencari judul buku. Fungsi *searching* yang saya gunakan adalah *binary search*.

*Binary Search* adalah cara membagi data menjadi 2 bagian sekali terjadi terjadi proses pencarian. Algoritma ini bekerja dengan membagi *dynamic array* menjadi dua bagian secara berulang, dan membandingkan nilai yang dicari dengan nilai tengah *dynamic array*. Pada kasus ini, nilai tersebut adalah judul buku.

Awalnya saya menggunakkan fungsi *binary search* sesuai dengan yang ada di dalam *powerpoint* yang diberikan oleh Pak Ricky. Tapi karena ketidaksesuaian tipe data, maka saya ubah sesuai dengan kebutuhan tipe data.

// PENGAPLIKASIAN SEARCH

int BinarySearchByTitle(vector<BookData>& CorneliusLibrary, string &TargetedTitle){ // fungsi untuk mencari judul menggunakkan binary search

    int Left = 0; // inisiasi batas kiri

    int Right = CorneliusLibrary.size() - 1; // inisiasi batas kanan

    while(Left <= Right){

        int Middle = (Left + Right) / 2; // pembagian tengah dynamic array

        if(CorneliusLibrary[Middle].Title == TargetedTitle) return Middle; // judul ditemukan

        if(CorneliusLibrary[Middle].Title < TargetedTitle) Left = Middle + 1; // judul yang dicari berada di sebelah kanan

        else Right = Middle - 1; // judul yang dicari berada di sebelah kiri

    }

    return -1; // jika judul tidak ditemukan

}

Berikut adalah hasil output dari *searching* berdasarkan data buku di atas

=========================

   Dashboard Librarian

=========================

1. Borrowing List

2. Insert new book

3. Sort books

4. Find book

5. Delete book

0. Exit

> 4

   1. Find book based on title

   2. Find book based on author

   > 1

Find the title

> Matahari

Book was Found

Code Book: 132016

Title: Matahari

Author: Tere Liye

Year: 2016

Press enter to continue

Tetapi, apabila hasil pencarian tidak ditemukan, maka berikut adalah outputnya

=========================

   Dashboard Librarian

=========================

1. Borrowing List

2. Insert new book

3. Sort books

4. Find book

5. Delete book

0. Exit

> 4

   1. Find book based on title

   2. Find book based on author

   > 1

Find the title

> C++ Coding

Book was not found

Press enter to continue

1. *Tree*

*Tree* adalah struktur data yang terdiri dari *root* dan *subtree* yang disusun secara berhirarki ke bawah. *Tree* pada program saya digunakan untuk menyimpan data buku yang sama dengan *vector*. Hanya saja, *tree* saya gunakan untuk melakukan pencarian berdasarkan nama *author* (sedangkan *vector* digunakan untuk melakukan pencarian berdasarkan judul buku).

Saya menggunakkan 7 dari 14 fungsi *tree* yang terdapat dalam *powerpoint* untuk memastikan efisiensi pemrograman, tetapi tidak mengorbankan fungsionalitas program (dimana program dapat berjalan dengan baik). Fungsi-fungsi tersebut adalah fungsi *Create*, untuk membuat *tree,* fungsi *InsertNode* untuk memasukkan *node* ke dalam *tree,* fungsi *DisplayInOrder* untuk menampilkan tree secara urut (berdasarkan judul)*,* fungsi *DeleteNode* untuk menghapus data buku dari *tree,* fungsi *FindMax* untuk mencari data buku paling ujung kanan (huruf depan paling terakhir) yang akan membantu proses penghapusan.

// PENGAPLIKASIAN TREE

void Create(TreeBook \*&Author){ // fungsi membuat tree

    Author = nullptr;

}

int IsEmpty(TreeBook \*Author){ // fungsi untuk memeriksa apakah tree kosong atau tidak

    if (Author == nullptr) return 1; // apabila kosong, maka mengembalikan nilai 1

    else return 0;

}

TreeBook\* InsertNode(TreeBook \*&Author,  BookData &Book){  // fungsi untuk menginputkan node ke dalam tree

    if(Author == nullptr){ //jika tree masih kosong

        TreeBook \*NewNode = new TreeBook; // maka membuat node pertama

        NewNode->Data = Book;

        NewNode->Left = nullptr;

        NewNode->Right = nullptr;

        Author = NewNode;

    }

    else if (Book.Title < Author->Data.Title){ // jika urutan huruf depan lebih kecil dari buku saat ini, maka melakukan pencarian ke kiri

        Author->Left = InsertNode(Author->Left, Book);

    }

    else if (Book.Title > Author->Data.Title){ // jika urutan huruf depan lebih besar dari buku saat ini, maka melakukan pencarian ke kanan

        Author->Right = InsertNode(Author->Right, Book);

    }

    return Author;

}

void DisplayInOrder(TreeBook \*Author){ // fungsi untuk menampilkan daftar buku secara urut berdasarkan huruf depan buku

    if(Author == nullptr) return; // jika tree masih kosong, maka mengembalikan nilai kosong

    DisplayInOrder(Author->Left); // melakukan penelusuran ke kiri

    cout << Author->Data.Title << " written by " << Author->Data.Author << endl; // menampilkan data buku

    cout << "released in " << Author->Data.Year << " with code book " << Author->Data.CodeBook << endl << endl;

    DisplayInOrder(Author->Right); // melakukan penelusuran ke kanan

}

TreeBook\* SearchAuthor(TreeBook \*Author, string &TargetedAuthor){ // fungsi untuk mencari author buku menggunakkan BST (Binary Search Tree)

    if(Author == nullptr) return nullptr;

    else if(TargetedAuthor < Author->Data.Author){ // jika urutan huruf depan lebih kecil dari tree, maka melakukan pencarian ke kiri

        SearchAuthor(Author->Left,TargetedAuthor);

    }

    else if(TargetedAuthor > Author->Data.Author){

        SearchAuthor(Author->Right,TargetedAuthor); // jika urutan huruf depan lebih kecil dari tree, maka melakukan pencarian ke kiri

    }

    return Author; // jika ditemukan, maka langsung mengembalikan data buku

}

TreeBook\* DeleteNode(TreeBook \*Author,string &TargetedTitle){ // fungsi untuk menghapus data buku

    if(Author == nullptr) return nullptr; // jika tree kosong, maka mengembalikan nilai kosong

    else if(TargetedTitle < Author->Data.Title){ // jika data yang dicari urutan hurufnya lebih kecil dari tree

        Author->Left = DeleteNode(Author->Left,TargetedTitle); // maka bergeser ke sebelah kiri

    }

    else if(TargetedTitle > Author->Data.Title){ // jika data yang dicari urutan hurufnya lebih kecil dari tree

        Author->Right = DeleteNode(Author->Right,TargetedTitle); // maka bergeser ke sebelah kanan

    }

    // jika data memang sudah ditemukan

    else{ // jika tree memiliki 1 child saja

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

            TreeBook \*Temp = Author->Right; // maka Temp akan menyimpan child sebelah kanan

            delete Author; // menghapus data

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

        }

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

            TreeBook \*Temp = Author->Left; // maka Temp akan menyimpan child sebelah kiri

            delete Author; // menghapus data

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

        }

        else{ //jika tree memiliki 2 child

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

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

            Author->Left = DeleteNode(Author->Left,Temp->Data.Title); // maka dilakukan penghapusan

        }

    }

    return Author; // mengembalikan tree

}

TreeBook \*FindMax(TreeBook \*Author){ // fungsi untuk mencari node paling ujung kanan (yang akan membantu fungsi DeleteNode)

    if(Author == nullptr) return nullptr; // jika tree kosong, maka mengembalikan nilai kosong

    else{

        if(Author->Right == nullptr) return Author; // Jika sudah tidak terdapat child di sebelah kanan, maka data paling ujung kanan telah ditemukan

        else return FindMax(Author->Right); // jika belum, maka pelakukan penelusuran ke sebelah kanan

    }

}

Berikut adalah hasil output dari *tree* berdasarkan data buku di atas

Book 1

    Automate the Boring Stuff with Python written by Al Sweigart

    released in 2015 with code book 001115

Book 2

    Bintang written by Tere Liye

    released in 2017 with code book 022017

Book 3

    Bulan written by Tere Liye

    released in 2015 with code book 022015

Book 4

    Bumi written by Tere Liye

    released in 2014 with code book 022014

Book 5

    C++: The Complete Reference written by Herbert Schildt

    released in 2015 with code book 003815

Book 6

    Ceros dan Batazoar written by Tere Liye

    released in 2018 with code book 032018

Book 7

    Clean Code written by Robert Cecil Martin

    released in 2008 with code book 003188

Book 8

    Competitive Programming: Algorithms and Techniques written by Cormen, Leiserson, Rivest, Stein

    released in 2009 with code book 000339

Book 9

    Fluent Python written by Luciano Ramalho

    released in 2015 with code book 061215

Book 10

    Komet written by Tere Liye

    released in 2018 with code book 112018

Book 11

    Komet Minor written by Tere Liye

    released in 2019 with code book 112019

Book 12

    Kotlin Programming: From Idea to Production written by JetBrains

    released in 2018 with code book 111018

Book 13

    Matahari written by Tere Liye

    released in 2016 with code book 132016

Book 14

    Nebula written by Tere Liye

    released in 2021 with code book 142021

Book 15

    Python Crash Course written by Eric Matthes

    released in 2015 with code book 016515

Book 16

    Selena written by Tere Liye

    released in 2020 with code book 192020

Book 17

    Si Putih written by Tere Liye

    released in 2022 with code book 192022

Book 18

    The Pragmatic Programmer written by Andy Hunt & Dave Thomas

    released in 1999 with code book 020199

Book 19

    The Rust Programming Language written by Steve Klabnik

    released in 2018 with code book 201918

Press Enter to continue

1. *Graph*

*Graph* adalah representasi objek-objek diskrit dan hubungan antara objek-objek tersebut. Pada program saya, *graph* saya gunakan untuk menghubungkan perekomendasian antara buku satu dengan buku yang lain. Jadi, terdapat 3 rekomendasi dalam program saya, yaitu serial tere liye, buku pembelajaran python, dan buku pembelajaran bahasa pemrogaraman lain

Graph TereLiyeRecommend;

Graph PythonRecommend;

Graph LearnOtherRecommend;

TereLiyeRecommend.Books = 10;

PythonRecommend.Books = 3;

LearnOtherRecommend.Books = 3;

AddLine(TereLiyeRecommend, "Bumi", "Bulan");

AddLine(TereLiyeRecommend, "Bulan", "Matahari");

AddLine(TereLiyeRecommend, "Matahari", "Bintang");

AddLine(TereLiyeRecommend, "Bintang", "Ceros dan Batazoar");

AddLine(TereLiyeRecommend, "Ceros dan Batazoar", "Komet");

AddLine(TereLiyeRecommend, "Komet", "Komet Minor");

AddLine(TereLiyeRecommend, "Komet Minor", "Selena");

AddLine(TereLiyeRecommend, "Selena", "Nebula");

AddLine(TereLiyeRecommend, "Nebula", "Si Putih");

AddLine(PythonRecommend, "Python Crash Course", "Automate the Boring Stuff with Python");

AddLine(PythonRecommend, "Python Crash Course", "Fluent Python");

AddLine(PythonRecommend, "Automate the Boring Stuff with Python", "Python Crash Course");

AddLine(PythonRecommend, "Automate the Boring Stuff with Python", "Fluent Python");

AddLine(PythonRecommend, "Fluent Python", "Python Crash Course");

AddLine(PythonRecommend, "Fluent Python", "Automate the Boring Stuff with Python");

AddLine(LearnOtherRecommend, "C++: The Complete Reference", "The Rust Programming Language");

AddLine(LearnOtherRecommend, "C++: The Complete Reference", "Kotlin Programming: From Idea to Production");

AddLine(LearnOtherRecommend, "The Rust Programming Language", "C++: The Complete Reference");

AddLine(LearnOtherRecommend, "The Rust Programming Language", "Kotlin Programming: From Idea to Production");

AddLine(LearnOtherRecommend, "Kotlin Programming: From Idea to Production", "C++: The Complete Reference");

AddLine(LearnOtherRecommend, "Kotlin Programming: From Idea to Production", "The Rust Programming Language");

Fungsi yang digunakan adalah fungsi *AddLine* untuk menambah *edge* antara satu buku dengan buku lain dan fungsi *DFS* untuk melakukan penelusuran rekomendasi buku.

// PENGAPLIKASIAN GRAPH

void AddLine(Graph &Recommendation, string From, string To){ // fungsi menambah edge ke dalam graph

    Recommendation.AdjList[From].push\_back(To); // terdapat garis penghubung antara buku satu dengan buku yang lain

}

void DFS(const Graph &Recommendation, const string &From){ // fungsi DFS (Depth First Search) pada graph menggunakkan stack

    unordered\_map<string,bool> Read; // membuat unordered\_map untuk melacak node/buku yang telah dikunjungi

    for(const auto &Pair : Recommendation.AdjList){ // inisialisasi semua node sebagai belum dikunjungi

        Read[Pair.first] = false;

    }

    stack<string> ToRead; // stack untuk melacak node yang akan dikunjungi

    ToRead.push(From); // masukkan node awal ke dalam stack

    while(!ToRead.empty()){ // selama stack tidak kosong

        string Current = ToRead.top();

        ToRead.pop(); // ambil node dari stack

        if(!Read[Current]){ // jika ada node/buku yang belum dikunjungi

            Read[Current] = true; // maka dikunjungi

            cout << Current << "  --->  "; // dan mengeluarkan node/buku saat ini

           auto GraphRec = Recommendation.AdjList.find(Current); // auto GraphRec untuk mengetahui secara pasti graph dari rekomendasi buku

            if (GraphRec != Recommendation.AdjList.end()) { // selama GraphRec belum mencapai akhir

                for (const string &Next : GraphRec->second) {

                    if (!Read[Next]) { // Jika ada tetangga yang belum dikunjungi, masukkan ke dalam stack

                        ToRead.push(Next);

                    }

                }

            }

        }

    }

}

Berikut adalah hasil output *graph*

We would also like to recommend you our suggested python learning course

Python Crash Course  --->  Fluent Python  --->  Automate the Boring Stuff with Python  --->  Finish

Or if you want to get acquainted with other programming language

C++: The Complete Reference  --->  Kotlin Programming: From Idea to Production  --->  The Rust Programming Language  --->  Finish

We would also like to recommend you Bumi novel series written by Tere Liye, a brilliant author from Indonesia

Bumi  --->  Bulan  --->  Matahari  --->  Bintang  --->  Ceros dan Batazoar  --->  Komet  --->  Komet Minor  --->  Selena  --->  Nebula  --->  Si Putih  --->  Finish

Thank you and have a nice day !!

1. *Table Hash* dan *Collision Handling*

*Table Hash* adalah struktur data yang menyimpan data menurut hasil *hash* (dengan melakukan *modulo* berdasarkan kode buku yang diinputkan) dan data disimpan ke dalam table berdasarkan hasil indeksnya.

*Collision Handling* adalah pengaturan sistematis di mana dua data yang berbeda tetapi memiliki hasil indeks *hash* yang sama. Dimana data dapat dimasukkan ke dalam index *table* yang sama tetapi tetap dikaitkan dengan data lain yang memiliki hasil *index* sama (*chaining*), atau data dapat diletakkan pada *table* berikutnya (*linear probing*).

Pada program saya, *Table Hash* digunakan untuk menyimpan data peminjaman buku. Kode buku di *hash* ing untuk mengetahui indeks mana data peminjaman dapat disimpan. *Collision Handling* yang saya gunakan adalah teknik *chaining*.

Terdapat fungsi *HashFunction* untuk melakukan *modulo* terhadap kode buku, fungsi *AddBook* untuk menginputkan data peminjaman buku ke dalam *table hash*, fungsi *FindBookBasedOnTitle*, dan fungsi *FindBookBasedOnAuthor* untuk mencari data buku berdasarkan judul buku atau penulis, fungsi *BorrowBookBasedOnTitle* dan fungsi *BorrowBookBasedOnAuthor* untuk melakukan peminjaman dan penghubung antara fungsi *FindBook* dan *AddBook*. Fungsi *DisplayBorrowList* untuk menampilkan semua data peminjaman.

// PENGAPLIKASIAN TABEL HASH DAN COLLISION HANDLING

int HashFunction(string &CodeBook){ // fungsi hashing

    int Hash = 0;

    for(char Code :CodeBook){

        Hash = (Hash % Code) % TableSize; // melakukan modulo pada kode buku, sehingga tahu dimana data buku diletakkan

    }

    return Hash;

}

void AddBook(BookData &Book){ // fungsi memasukkan data buku ke dalam table hash

    int i = HashFunction(Book.CodeBook); // hashing kode buku

    BookTable[i].push\_back({Book,"", ""}); // memasukkan data buku ke dalam table hash

}

BookData\* FindBookBasedOnTitle(string &TargetedTitle){ // fungsi mencari buku berdasarkan judul

    int i = HashFunction(TargetedTitle); // hashing judul

    for(auto &FindBook :BookTable[i]){ // pencarian judul buku

        if(FindBook.Book.Title == TargetedTitle){

            return &FindBook.Book; // jika memang ditemukan, maka mengembalikan data buku tersebut

        }

    }

    return nullptr; // jika memang tidak ditemukan, maka mengembalikan pointer kosong

}

void BorrowBookBasedOnTitle(string &TargetedTitle, string &BorrowerName, string &BorrowDate){ // fungsi meminjam buku berdasarkan judul

    BookData \*FindBook = FindBookBasedOnTitle(TargetedTitle); // melakukan pencarian judul dengan fungsi FindBookBasedOnTitle

    if(FindBook != nullptr){ // jika data buku ditemukan

        BorrowerData InputNewData = {\*FindBook, BorrowerName, BorrowDate}; // menyimpan data buku, nama peminjam dan tanggal hari ini

        int i = HashFunction(TargetedTitle); // melakukan proses hashing untuk mengetahui pada indeks berapa data masuk

        BorrowTable[i].push\_back(InputNewData); // memasukkan data peminjam ke dalam tabel hashing

    }

    else{

        cout << "Book was not found or already borrowed\n";

    }

}

BookData\* FindBookBasedOnAuthor(string &TargetedAuthor){ // fungsi mencari buku berdasarkan author

    int i = HashFunction(TargetedAuthor); // hashing judul

    for(auto &FindBook :BookTable[i]){  // pencarian author buku

        if(FindBook.Book.Author == TargetedAuthor){

            return &FindBook.Book; // jika memang ditemukan, maka mengembalikan data buku tersebut

        }

    }

    return nullptr; // jika memang tidak ditemukan, maka mengembalikan pointer kosong

}

void BorrowBookBasedOnAuthor(string &TargetedAuthor, string &BorrowerName, string &BorrowDate){ // fungsi meminjam buku berdasarkan author

    BookData \*FindBook = FindBookBasedOnAuthor(TargetedAuthor); // melakukan pencarian judul dengan fungsi FindBookBasedOnAuthpr

    if(FindBook != nullptr){ // jika data buku ditemukan

        BorrowerData InputNewData = {\*FindBook, BorrowerName, BorrowDate};  // menyimpan data buku, nama peminjam dan tanggal hari ini

        int i = HashFunction(TargetedAuthor); // melakukan proses hashing untuk mengetahui pada indeks berapa data masuk

        BorrowTable[i].push\_back(InputNewData); // memasukkan data peminjam ke dalam tabel hashing

    }

    else{

        cout << "Book was not found or already borrowed\n";

    }

}

void DisplayBorrowList(){ // fungsi untuk menampilkan semua data peminjaman

    bool IsEmpty = true;

    for(int i = 0;i < TableSize;i++){

        if(!BorrowTable[i].empty()){

            IsEmpty == false;

            break;

        }

    }

    if(IsEmpty){ // jika memang kosong, maka program akan memberitahu

        cout << "\nNo borrowing process were made\n";

        return;

    }

    else{ // jika memang ada, maka menampilkan semua data

        for(int i = 0;i < TableSize; i++){

            for(auto &j : BorrowTable[i]){

                cout << "\n| " << j.Book.Title << " |" << " written by " << " | " <<j.Book.Author << " |,\n"; // menampilkan data buku dan data penulis

                cout << "borrowed by " << j.BorrowerName << " from " << j.BorrowDate << endl << endl; // menampilkan nama penulis dan tanggal peminjaman

            }

        }

    }

}

Berikut adalah hasil output *table hash*

=========================

   Dashboard Librarian

=========================

1. Borrowing List

2. Insert new book

3. Sort books

4. Find book

5. Delete book

0. Exit

> 1

| Clean Code | written by | Robert Cecil Martin |

   borrowed by Louis from 19-05-2024

Press enter to continue

Fungsi lain yang belum dijelaskan :

1. Fungsi *void InputBookData* untuk menginputkan data buku baru ke dalam *dynamic array*

void InputBookData(BookData& Book){ // fungsi memasukkan data buku baru

    cout << "\nInsert title of the Book\n";

    cout << "> ";

    getline(cin, Book.Title);

    cout << endl;

    cout << "Insert author of the Book\n";

    cout << "> ";

    getline(cin, Book.Author);

    cout << endl;

    cout << "Insert year released of the Book\n";

    cout << "> ";

    cin >> Book.Year;

    cin.ignore();

    Book.CodeBook = GenerateCodeBook(Book.Title,Book.Author,Book.Year); // membuat kode buku berdasarkan judul, penulis, tahun

    cout << "\nCode Book: " << Book.CodeBook << endl;

    cout << "Title: " << Book.Title << endl;

    cout << "Author: " << Book.Author << endl;

    cout << "Year: " << Book.Year << endl << endl;

    cout << Book.Title << " has been inputted into database\n";

}

1. Fungsi *string GenerateCodeBook* untuk membuat kode buku di perpus berdasarkan huruf pertama judul buku, huruf pertama

string GenerateCodeBook(string Title,string Author,int Year){ // fungsi membuat kode buku di perpus

    // mengambil huruf depan dari Title dan Author

    char FirstTitleChar = Title[0];

    char FirstAuthorChar = Author[0];

    // mengubah huruf depan menjadi angka

    int FirstTitleValue = FirstTitleChar - 'A' + 1;

    int FirstAuthorValue = FirstAuthorChar - 'A' + 1;

    // mengambil 2 digit terakhir dari tahun rilis (2 angka terakhir)

    int Last2DigitYear = Year % 100;

    // 2 angka awal diambil dari huruf awal judul

    // 2 angka tengah diambil dari huruf awal author

    // 2 angka terakhir diambil dari digit 2 tahun terakhir perilisan

    string GeneratedCode = to\_string(FirstTitleValue) + to\_string(FirstAuthorValue) + to\_string(Last2DigitYear); // penggabungan 1 kode buku,

    while (GeneratedCode.length() < 6){ // kode harus memiliki panjang 6 char

        GeneratedCode = '0' + GeneratedCode;

    }

    return GeneratedCode;

}

1. Fungsi *DeleteBookData* untuk menghapus data buku dari *dynamic array*

void DeleteBookData(vector<BookData>& Book, string &TargetedTitle){ // fungsi menghapus data buku

    for(size\_t i = 0; i < Book.size(); i++){ // iterasi pencarian judul buku yang akan dihapus

        if(Book[i].Title == TargetedTitle){ // jika telah ditemukan

            Book.erase(Book.begin() + i); // maka dilakukan proses penghapusan

            cout << endl << TargetedTitle << " has been deleted from database\n";

            return;

        }

    }

    cout << endl << TargetedTitle << " was not found\n"; // jika memang tidak ditemukan, maka tidak terjadi penghapusan

}

1. Fungsi *string GetCurrentDate* untuk mendapatkan tanggal hari ini dan membantu proses peminjaman data (dimana pengguna tidak perlu menginputkan tanggal hari ini)

string GetCurrentDate(){ // fungsi untuk mendapatkan tanggal hari ini

    time\_t Now = time(0); // inisiasi waktu sekarang

    tm \*LocalTime = localtime(&Now); // inisiasi waktu lokal

    string Day = to\_string(LocalTime->tm\_mday); // mendapatkan tanggal hari ini lalu diubah menjadi string

    string Month = to\_string(1 + LocalTime->tm\_mon); // mendapatkan nomor bulan saat ini lalu diubah menjadi string

    string Year = to\_string(1900 + LocalTime->tm\_year); // mendapatkan tahun saat ini lalu diubah menjadi string

    if (Day.length() < 2) Day = "0" + Day; // jika tanggal hari ini hanya terdiri atas 1 angka, ditambahkan 0 di depan

    if (Month.length() < 2) Month = "0" + Month;  // jika nomor bulan saat ini hanya terdiri atas 1 angka, ditambahkan 0 di depan

    return Day + "-" + Month + "-" + Year;

}

Penjelasan Program

Pada awal program utama, program akan melakukan inisiasi *graph* untuk melakukan rekomendasi dan menginputkan data buku *dynamic array* ke dalam *tree* dan *table hash*.

TereLiyeRecommend.Books = 10;

PythonRecommend.Books = 3;

LearnOtherRecommend.Books = 3;

AddLine(TereLiyeRecommend, "Bumi", "Bulan");

AddLine(TereLiyeRecommend, "Bulan", "Matahari");

AddLine(TereLiyeRecommend, "Matahari", "Bintang");

AddLine(TereLiyeRecommend, "Bintang", "Ceros dan Batazoar");

AddLine(TereLiyeRecommend, "Ceros dan Batazoar", "Komet");

AddLine(TereLiyeRecommend, "Komet", "Komet Minor");

AddLine(TereLiyeRecommend, "Komet Minor", "Selena");

AddLine(TereLiyeRecommend, "Selena", "Nebula");

AddLine(TereLiyeRecommend, "Nebula", "Si Putih");

AddLine(PythonRecommend, "Python Crash Course", "Automate the Boring Stuff with Python");

AddLine(PythonRecommend, "Python Crash Course", "Fluent Python");

AddLine(PythonRecommend, "Automate the Boring Stuff with Python", "Python Crash Course");

AddLine(PythonRecommend, "Automate the Boring Stuff with Python", "Fluent Python");

AddLine(PythonRecommend, "Fluent Python", "Python Crash Course");

AddLine(PythonRecommend, "Fluent Python", "Automate the Boring Stuff with Python");

AddLine(LearnOtherRecommend, "C++: The Complete Reference", "The Rust Programming Language");

AddLine(LearnOtherRecommend, "C++: The Complete Reference", "Kotlin Programming: From Idea to Production");

AddLine(LearnOtherRecommend, "The Rust Programming Language", "C++: The Complete Reference");

AddLine(LearnOtherRecommend, "The Rust Programming Language", "Kotlin Programming: From Idea to Production");

AddLine(LearnOtherRecommend, "Kotlin Programming: From Idea to Production", "C++: The Complete Reference");

AddLine(LearnOtherRecommend, "Kotlin Programming: From Idea to Production", "The Rust Programming Language");

for(auto& i : CorneliusLibrary){

    string GeneratingCodeBook= GenerateCodeBook(i.Title,i.Author,i.Year);

    i.CodeBook = GeneratingCodeBook;

}

Create(Author);

for(auto& i : CorneliusLibrary){

    InsertNode(Author,i);

}

for(auto& i:CorneliusLibrary){

    AddBook(i);

}

Setelah itu, program akan masuk ke dashboard awal, dimana terdapat pembagian 2 menu, yaitu menu khusus *librarian* dan menu khusus *visitor*.

   int Login;

    char Home = 'y';

    while(Home == 'y'){

        system("cls");

        cout << "=========================\n";

        cout << "     Cornelius Library   \n";

        cout << "=========================\n";

        cout << "1. Login as Librarian\n";

        cout << "2. Login as Visitor\n";

        cout << "0. Exit\n";

        cout << "> ";

        cin >> Login;

        cin.ignore();

Jika kita mengakses menu khusus *librarian*, maka program akan menyediakan 5 pilihan menu

switch (Login){

        case 1:{

            char LoginLibrarian = 'y';

            while (LoginLibrarian == 'y'){

                int Input;

                cout << "\n=========================\n";

                cout << "   Dashboard Librarian   \n";

                cout << "=========================\n";

                cout << "1. Borrowing List\n";

                cout << "2. Insert new book\n";

                cout << "3. Sort books\n";

                cout << "4. Find book\n";

                cout << "5. Delete book\n";

                cout << "0. Exit\n";

                cout << "> ";

Pada menu 1, akan menampilkan daftar peminjam buku (dengan penerapan *table hash*), menu 2 berfungsi untuk menginputkan data buku baru ke dalam *dynamic array*¸*tree* dan *table hash*. Menu 3 berfungsi untuk mengurukan buku berdasarkan urutan huruf awal pertamanya. Menu 4 berfungsi untuk mencari buku berdasarkan judul buku atau nama *author,* bergantung pada input pengguna, dan menu 5 berfungsi untuk menghapus data suatu buku. Menu 0 digunakan untuk keluar dari menu khusus *librarian*.

switch(Input){

    case 1:{

      char ManageBorrow;

      string TargetedCodeBook;

      string TargetedBorrowerName;

      DisplayBorrowList();

cout << "Press enter to continue\n";

      cin.get();

    }

     break;

     case 2:{

        char Choice = 'y';

          while(Choice == 'y'){

                BookData Book;

                InputBookData(Book);

                  CorneliusLibrary.push\_back(Book);

                  Author = InsertNode(Author,Book);

                  AddBook(Book);

cout << "Insert another book? (y/n)\n";

                  cout << "> ";

                  cin >> Choice;

                  cin.ignore();

            }

      }

       break;

       case 3:{

           SortBooksByTitle(CorneliusLibrary);

           cout << "List of the sorted books based on title\n\n";

           for(size\_t i = 0; i < CorneliusLibrary.size(); i++){

                 cout << "Book " << i + 1 << endl;

                 cout << "    | " << CorneliusLibrary[i].Title << " | written by | " << CorneliusLibrary[i].Author << " |\n";

                 cout << "    released in " <<  CorneliusLibrary[i].Year << " with code book " << CorneliusLibrary[i].CodeBook << "\n\n";

                        }

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

            cin.get();

            }

      break;

         case 4:{

              int InputFind;

              cout << "   1. Find book based on title\n";

              cout << "   2. Find book based on author\n";

              cout << "   > ";

              cin >> InputFind;

              cin.ignore();

             switch(InputFind){

                 case 1:{

                string TargetedTitle;

                  SortBooksByTitle(CorneliusLibrary);

                  cout << "\nFind the title\n";

                  cout << "> ";

                  getline(cin, TargetedTitle);

                  int FoundTitle = BinarySearchByTitle(CorneliusLibrary, TargetedTitle);

                  if(FoundTitle != -1){

                      cout << "\nBook was Found\n\n";

                      cout << "Code Book: " << CorneliusLibrary[FoundTitle].CodeBook << endl;

                      cout << "Title: " << CorneliusLibrary[FoundTitle].Title << endl;

                      cout << "Author: " << CorneliusLibrary[FoundTitle].Author << endl;

                      cout << "Year: " << CorneliusLibrary[FoundTitle].Year << endl;

                 }

                 else{

                     cout << "\nBook was not found\n";

                 }

           }

           break;

                case 2:{

                     string TargetedAuthor;

                     cout << "\nFind the author\n";

                       cout << "> ";

                       getline(cin, TargetedAuthor);

                       TreeBook \*FoundAuthor = SearchAuthor(Author,TargetedAuthor);

                       if(FoundAuthor != nullptr){

                             cout << "\nBook was found\n\n";

                             cout << "Code Book: " << FoundAuthor->Data.CodeBook << endl;

                             cout << "Title: " << FoundAuthor->Data.Title << endl;

                             cout << "Author: " << FoundAuthor->Data.Author << endl;

                             cout << "Year: " << FoundAuthor->Data.Year << endl;

                        }

                        else{

                             cout << "\nBook was not found\n";

                        }

                   }

                  break;

                   default:

                   break;

                   }

                    cout << "\nPress enter to continue\n";

                    cin.get();

                }

                break;

                case 5:{

                    string TargetedTitle;

                    cout << "\nInput the title\n";

                    cout << "> ";

                    getline(cin, TargetedTitle);

                    DeleteBookData(CorneliusLibrary,TargetedTitle);

                    DeleteNode(Author,TargetedTitle);

                    cout << "\nPress enter to continue\n";

                    cin.get();

                }

                break;

                case 0:

                    char Exit;

                    cout << "\nExit librarian mode? (y/n)\n";

                    cout << "> ";

                    cin >> Exit;

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

                        LoginLibrarian = 'n';

                    }

                    else{

                        LoginLibrarian = 'y';

                    }

                break;

                default:

                break;

            }

        }

    }

    break;

Jika kita mengakses menu *visitor*, maka program akan menyediakan 2 pilihan menu

char LoginVisitor = 'y';

        while (LoginVisitor == 'y'){

            int Input;

            cout << "\n===================================\n";

            cout << "    Welcome to Cornelius Library   \n";

            cout << "====================================\n";

            cout << "What can we assist you today?\n\n";

            cout << "1. See all of our collection\n";

            cout << "2. Find the book you want to borrow\n";

            cout << "0. Exit\n";

            cout << "> ";

            cin >> Input;

            cin.ignore();

Menu 1 untuk menampilkan semua koleksi buku yang dimiliki oleh perpustakaan, menu 2 untuk mencari buku yang diinginkan. Pengguna bisa memilih apakah ingin mencari buku berdasarkan judul atau nama penulisnya. Setiap kali kita mencari buku, program akan menawarkan kepada pengguna apakah pengguna ingin meminjam buku atau tidak. Jika buku tersebut dipinjam, maka proses peminjaman akan dilakukan (penerapan *table hash*). Menu 0 digunakan untuk keluar dari aplikasi.

switch (Input){

     case 1:{

        SortBooksByTitle(CorneliusLibrary);

        cout << "List of the sorted books based on title\n\n";

        for(size\_t i = 0; i < CorneliusLibrary.size(); i++){

            cout << "Book " << i + 1 << endl;

            cout << "    | " << CorneliusLibrary[i].Title << " | written by | " << CorneliusLibrary[i].Author << " |\n";

            cout << "    released in " <<  CorneliusLibrary[i].Year << " with code book " << CorneliusLibrary[i].CodeBook << "\n\n";

         }

         cout << "\nPress enter to continue\n";

        cin.get();

      }

      break;

       case 2:{

          int InputFind;

          cout << "   1. Find the book based on title\n";

          cout << "   2. Find the book based on author\n";

          cout << "   > ";

          cin >> InputFind;

          cin.ignore();

          switch(InputFind){

                case 1:{

                    char BorrowBook;

                    string TargetedTitle;

                    SortBooksByTitle(CorneliusLibrary);

                    cout << "\nWhat title are you looking for \n";

                    cout << "> ";

                     getline(cin, TargetedTitle);

                     int FoundTitle = BinarySearchByTitle(CorneliusLibrary, TargetedTitle);

                      if(FoundTitle != -1){

                        cout << "\nWe found the title you are looking for!!!\n\n";

                         cout << "Code Book: " << CorneliusLibrary[FoundTitle].CodeBook << endl;

                         cout << "Title: " << CorneliusLibrary[FoundTitle].Title << endl;

                         cout << "Author: " << CorneliusLibrary[FoundTitle].Author << endl;

                         cout << "Year: " << CorneliusLibrary[FoundTitle].Year << endl;

                         cout << "\nWould you like to borrow this book? (y/n)\n";

                         cout << "> ";

                         cin >> BorrowBook;

                         cin.ignore();

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

                              string BorrowerName;

                              string BorrowDate = GetCurrentDate();

                              cout << "\nWhat is your name? (We would like to manage your borrowing permission)\n";

                              cout << "> ";

                              getline(cin, BorrowerName);

                              BorrowBookBasedOnTitle(TargetedTitle,BorrowerName,BorrowDate);

                              cout << "\nYour borrowing permission has been processed, please proceed to the front office\n";

                              cout << "so you can bring the book with you as soon as posibble\n";

                              cout << "\nWe would also like to recommend you our suggested python learning course\n";

                            DFS(PythonRecommend, "Python Crash Course");

                              cout << "Finish\n";

                              cout << "\nOr if you want to get acquainted with other programming language\n";

                            DFS(LearnOtherRecommend, "C++: The Complete Reference");

                              cout << "Finish\n";

                              cout << "\nThank you and have a nice day !!\n";

                    }

                }

                     else{

                        cout << "\nUnfortunately, we did not find the title you are looking for...\n";

                     }

                cout << "\nPress enter to continue\n";

                cin.get();

                }

                break;

                case 2:{

                     char BorrowBook;

                     string TargetedAuthor;

                     cout << "\nWho is the author you are looking for\n";

                     cout << "> ";

                     getline(cin, TargetedAuthor);

                     TreeBook \*FoundAuthor = SearchAuthor(Author,TargetedAuthor);

                      if(FoundAuthor != nullptr){

                           cout << "\nWe found the author you are looking for!!!\n\n";

                           cout << "Code Book: " << FoundAuthor->Data.CodeBook << endl;

                           cout << "Title: " << FoundAuthor->Data.Title << endl;

                           cout << "Author: " << FoundAuthor->Data.Author << endl;

                           cout << "Year: " << FoundAuthor->Data.Year << endl;

                           cout << "Would you like to borrow this book? (y/n)\n";

                           cout << "> ";

                           cin >> BorrowBook;

                           cin.ignore();

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

                               string BorrowerName;

                               string BorrowDate = GetCurrentDate();

                               cout << "What is your name? (We would like to manage your borrowing permission) (y/n) \n";

                               cout << "> ";

                               getline(cin, BorrowerName);

                               BorrowBookBasedOnTitle(TargetedAuthor,BorrowerName,BorrowDate);

                               cout << "Your borrowing permission has been processed, please proceed to the front office\n";

                               cout << "so you can bring your book as soon as posibble\n";

                               cout << "\nWe would also like to recommend you Bumi novel series written by Tere Liye, a brilliant author from Indonesia\n";

                               DFS(TereLiyeRecommend, "Bumi");

                               cout << "Finish\n";

                               cout << "Thank you and have a nice day !!\n";

                           }

                        }

                        else{

                          cout << "\nUnfortunately, we did not find the author you are looking for...\n";

                        }

                        cout << "\nPress enter to continue\n";

                        cin.get();

                        }

                        break;

                    default:

                    break;

                    }

            }

            break;

            case 0:{

                char Exit;

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

                cout << "> ";

                cin >> Exit;

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

                    cout << "Thank you for visiting Cornelius Library\n";

                    cout << "Have a nice day !! \n";

                    LoginVisitor = 'n';

                }

                else{

                    LoginVisitor = 'y';

                }

            }

            break;

            default:

            break;

            }

        }

    }

    break;

Setelah proses peminjaman, program akan menampilkan rekomendasi buku yang dapat dibaca oleh pengguna (penerapan *graph*).

Jika pengguna ingin keluar dari aplikasi, maka pengguna dapat memilih menu 0 untuk keluar dari aplikasi *Library.*

    case 0:{

        char Exit;

        cout << "\nExit library? (y/n)\n";

        cout << "> ";

        cin >> Exit;

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

                Home = 'n';

            }

            else{

                Home = 'y';

            }

        }

    break;

    default:

    break;

    }

    }

    return 0;

}