LAPORAN TUGAS BESAR

IF2110/Algoritma dan Struktur Data

Engi’s Kitchen Expansion

Dipersiapkan oleh:

Kelompok 11 – Warteg Bahari

- 13517004 Bimo Adityarahman W.

- 13517025 Ricky Yuliawan

- 13517058 Ahmad Rizqee Nurhani

- 13517064 Naufal Aditya D.

- 13517076 Ahmad Rizal Alifio

Sekolah Teknik Elektro dan Informatika - Institut Teknologi Bandung

Jl. Ganesha 10, Bandung 40132

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Sekolah Teknik Elektro dan Informatika ITB** | Nomor Dokumen | | Halaman |
| *IF2110-TB-11-K01* | | *<jml hlm>* |
| *Revisi* | *<no revisi>* | *<Tgl release>* |

Daftar Isi

[1 Ringkasan 3](#_Toc244931066)

[2 Penjelasan Tambahan Spesifikasi Tugas 3](#_Toc244931067)

[2.1 Randomizer 3](#_Toc244931068)

[2.2 Menampilkan Pintu 3](#_Toc244931069)

[3 Struktur Data 4](#_Toc244931070)

[3.1 <Struktur Data 1> 3](#_Toc244931071)

[3.2 <Struktur Data 2> 3](#_Toc244931072)

[4 Program Utama 3](#_Toc244931073)

[5 Algoritma-Algoritma Menarik 4](#_Toc244931074)

[5.1 <Algoritma 1> 4](#_Toc244931075)

[5.2 <Algoritma 2> 4](#_Toc244931076)

[6 Data Test 4](#_Toc244931077)

[6.1 <Data Test 1> 4](#_Toc244931078)

[6.2 <Data Test 2> 4](#_Toc244931079)

[7 Test Script 4](#_Toc244931080)

[8 Pembagian Kerja dalam Kelompok 5](#_Toc244931081)

[9 Lampiran 5](#_Toc244931082)

[9.1 Deskripsi Tugas Besar 2 5](#_Toc244931083)

[9.2 Notulen Rapat 5](#_Toc244931084)

[9.3 Log Activity Anggota Kelompok 5](#_Toc244931085)

[9.4 <Lain-Lain> 5](#_Toc244931086)

# Ringkasan

Restoran (rumah makan) adalah usaha dalam bidang tata boga yang menyajikan hidangan kepada masyarakat dan menyediakan tempat untuk menikmati hidangan tersebut serta menetapkan tarif tertentu untuk makanan dan pelayanannya, tetapi ada juga yang menyediakan layanan *take-out dining* dan *delivery service* sebagai salah satu bentuk pelayanan kepada konsumennya. Dalam zaman teknologi yang sudah maju ini, ada yang namanya program simulasi, yaitu program perancangan suatu obyek diam/bergerak dengan parameter yang mendekati nilai sebenarnya. Program simulasi restoran ini adalah program perancangan restoran dimana keberjalanan programnya hampir mendekati sebagaimana mestinya menjalankan suatu usaha restoran. Dalam program ini diminta untuk memanjemen restoran. Restoran memiliki ruangan-ruangan, berupa ruang untuk pengunjung, yang berisi kursi dan meja untuk pengunjung, dan dapur, yang merupakan tempat untuk membuat makanan yang dibuat. Pengunjung yang mendatangi restoran akan mengantre terlebih dahulu. Pemain lalu akan mengarahkan pengunjung yang berada pada antrean paling depan untuk duduk di meja yang cukup kapasitasnya. Setelah pengunjung duduk di meja, mereka akan langsung melakukan pemesanan makanan. Pengunjung yang sedang mengantre diharapkan menunggu, jika pengunjung tersebut menunggu lebih lama maka pengunjung akan pulang dan kredibilitas restoran berkurang. Pembuatan makanan dilakukan dengan menambahkan bahan makanan kepada piring. Pencampuran bahan makanan mengikuti urutan tertentu berdasarkan pohon makanan. Makanan yang telah terbentuk diletakkan di nampan terlebih dahulu sebelum dibawa ke pengunjung.

Di dalam laporan ini dijelaskan deskripsi persoalan secara singkat, pemaparan asumsi-asumsi yang diambil dalam pembuatan Tugas Besar (beserta spesifikasi fitur-fitur tambahan yang digunakan), pemaparan mengenai struktur data yang digunakan, penjelasan mengenai langkah kerja program utama, rincian *data-test* yang digunakan selama sesi *testing*, serta tabel pembagian tugas dalam kelompok.

Program yang kami buat sudah memenuhi semua spesifikasi yang wajib dikerjakan menurut spesifikasi yang diberikan. Program kami juga bisa menyimpan permainan jika permainan yang sedang dijalankan belum selesai. Setelah itu, file yang disimpan bisa digunakan untuk dimuat pada saat dibutuhkan.

# Penjelasan Tambahan Spesifikasi Tugas

## Randomizer

Randomizer merupakan fungsi untuk men-*generate* objek pada program simulasi restoran secara random, salah satunya pada pengunjung. Fungsi ini akan memunculkan pengunjung ke restoran secara random, serta akan menginput kesabaran pengunjung untuk menunggu pesanan mereka secara random.

## Menampilkan Pintu

Menampilkan pintu merupakan menampilkan objek pintu pada peta dengan legenda karakter “D” dimana jika dilewati player akan berganti ruangan. Jadi, jika player ingin berpindah ke ruangan lain, harus melewati objek pintu tersebut.

# Struktur Data (ADT)

## Array Pesanan

Pada ADT Array Pesanan (file header : arraypesanan.h, file body : arraypesanan.c), kami menggunakan struktur data array. Alasan kami memilih array adalah karena pada program simulasi restoran, diperlukan untuk representasi pesanan ​(order). Ada beberapa tipe buatan pada ADT Array Pesanan, yaitu PesananID dan TabPesanan. PesananID adalah tipe yang digunakan untuk menyimpan informasi nomor pesanan pengunjung dan informasi kondisi menunggunya pengunjung. Kode programnya dapat dilihat sebagai berikut.

typedef struct {

Kata Menu;

int MejaID;

int Kesabaran;

} PesananID;

Dari kode diatas dapat dilihat bahwa kita membutuhkan ADT lain untuk menyusun tipe buatan seperti tipe Kata dan boolean.

Selanjutnya adalah TabPesanan adalah tipe yang digunakan untuk menyimpan pesanan para pengunjung, seperti nomor meja dan makanan yang mereka pesan. Kode programnya dapat dilihat sebagai berikut

typedef struct {

PesananID TI[IdxMax+1];

int Neff;

} TabPesanan;

ADT Array Pesanan menyelesaikan beberapa permasalahan yaitu menyimpan data pesanan para pengunjung, kondisi pengunjung menunggu pesanan mereka, dan nomor meja para pengunjung.

Beberapa primitif yang kami gunakan pada ADT ini adalah

|  |
| --- |
| void MakeEmptyArray (TabPesanan \* T);  /\* I.S. T sembarang \*/  /\* F.S. Terbentuk tabel T kosong dengan kapasitas IdxMax-IdxMin+1 \*/  int NbElmtArray (TabPesanan T);  /\* Mengirimkan banyaknya elemen efektif tabel \*/  /\* Mengirimkan nol jika tabel kosong \*/  /\* \*\*\* Daya tampung container \*\*\* \*/  int MaxNbEl (TabPesanan T);  /\* Mengirimkan maksimum elemen yang dapat ditampung oleh tabel \*/  /\* \*\*\* Selektor INDEKS \*\*\* \*/  int GetFirstIdx (TabPesanan T);  /\* Prekondisi : Tabel T tidak kosong \*/  /\* Mengirimkan indeks elemen T pertama \*/  int GetLastIdx (TabPesanan T);  /\* Prekondisi : Tabel T tidak kosong \*/  /\* Mengirimkan indeks elemen T terakhir \*/  boolean IsIdxValidArray (TabPesanan T, int i);  /\* Mengirimkan true jika i adalah indeks yang valid utk ukuran tabel \*/  /\* yaitu antara indeks yang terdefinisi utk container\*/  boolean IsIdxEffArray (TabPesanan T, int i);  /\* Mengirimkan true jika i adalah indeks yang terdefinisi utk tabel \*/  /\* yaitu antara FirstIdx(T)..LastIdx(T) \*/  boolean IsEmptyArray (TabPesanan T);  /\* Mengirimkan true jika tabel T kosong, mengirimkan false jika tidak \*/  boolean IsFullArray (TabPesanan T);  /\* Mengirimkan true jika tabel T penuh, mengirimkan false jika tidak \*/  void BacaIsi (TabPesanan \* T);  /\* I.S. T sembarang \*/  /\* F.S. Tabel T terdefinisi \*/  /\* Proses : membaca banyaknya elemen T dan mengisi nilainya \*/  /\* 1. Baca banyaknya elemen diakhiri enter, misalnya N \*/  /\* Pembacaan diulangi sampai didapat N yang benar yaitu 0 <= N <= MaxNbEl(T) \*/  /\* Jika N tidak valid, tidak diberikan pesan kesalahan \*/  /\* 2. Jika 0 < N <= MaxNbEl(T); Lakukan N kali: Baca elemen mulai dari indeks  IdxMin satu per satu diakhiri enter \*/  /\* Jika N = 0; hanya terbentuk T kosong \*/  void BacaIsiTab (TabPesanan \* T);  /\* I.S. T sembarang \*/  /\* F.S. Tabel T terdefinisi \*/  /\* Proses : membaca elemen T sampai dimasukkan nilai -9999 \*/  /\* Dibaca elemen satu per satu dan disimpan mulai dari IdxMin \*/  /\* Pembacaan dihentikan jika pengguna memasukkan nilai -9999 \*/  /\* Jika dari pertama dimasukkan nilai -9999 maka terbentuk T kosong \*/  void TulisIsi (TabPesanan T);  /\* Proses : Menuliskan isi tabel dengan traversal \*/  /\* I.S. T boleh kosong \*/  /\* F.S. Jika T tidak kosong : indeks dan elemen tabel ditulis berderet ke bawah \*/  /\* Jika T kosong : Hanya menulis "Tabel kosong" \*/  /\* Contoh: Jika isi Tabel: [1, 20, 30, 50]  Maka tercetak di layar:  [1]1  [2]20  [3]30  [4]50  \*/  void TulisIsiTab (TabPesanan T);  /\* Proses : Menuliskan isi tabel dengan traversal, tabel ditulis di antara kurung siku;  antara dua elemen dipisahkan dengan separator "koma", tanpa tambahan karakter di depan, di tengah, atau di belakang, termasuk spasi dan enter \*/  /\* I.S. T boleh kosong \*/  /\* F.S. Jika T tidak kosong: [e1,e2,...,en] \*/  /\* Contoh : jika ada tiga elemen bernilai 1, 20, 30 akan dicetak: [1,20,30] \*/  /\* Jika tabel kosong : menulis [] \*/  boolean SearchArray (TabPesanan T, PesananID X);  /\* Search apakah ada elemen tabel T yang bernilai X \*/  /\* Jika ada, menghasilkan true, jika tidak ada menghasilkan false \*/  /\* Memakai Skema search DENGAN boolean \*/  void CopyTab (TabPesanan Tin, TabPesanan \* Tout);  /\* I.S. Tin terdefinisi, Tout sembarang \*/  /\* F.S. Tout berisi salinan dari Tin (elemen dan ukuran identik) \*/  /\* Proses : Menyalin isi Tin ke Tout \*/  TabPesanan InverseTab (TabPesanan T);  /\* Menghasilkan tabel dengan urutan tempat yang terbalik, yaitu : \*/  /\* elemen pertama menjadi terakhir, \*/  /\* elemen kedua menjadi elemen sebelum terakhir, dst.. \*/  /\* Tabel kosong menghasilkan tabel kosong \*/  boolean IsSimetris (TabPesanan T);  /\* Menghasilkan true jika tabel simetrik \*/  /\* Tabel disebut simetrik jika: \*/  /\* elemen pertama = elemen terakhir, \*/  /\* elemen kedua = elemen sebelum terakhir, dan seterusnya \*/  /\* Tabel kosong adalah tabel simetris \*/  void AddAsLastEl (TabPesanan \* T, PesananID X);  /\* Proses: Menambahkan X sebagai elemen terakhir tabel \*/  /\* I.S. Tabel T boleh kosong, tetapi tidak penuh \*/  /\* F.S. X adalah elemen terakhir T yang baru \*/  void AddEli (TabPesanan \* T, PesananID X, int i);  /\* Menambahkan X sebagai elemen ke-i tabel tanpa mengganggu kontiguitas  terhadap elemen yang sudah ada \*/  /\* I.S. Tabel tidak kosong dan tidak penuh \*/  /\* i adalah indeks yang valid. \*/  /\* F.S. X adalah elemen ke-i T yang baru \*/  /\* Proses : Geser elemen ke-i+1 s.d. terakhir \*/  /\* Isi elemen ke-i dengan X \*/  void DelLastEl (TabPesanan \* T, PesananID \* X);  /\* Proses : Menghapus elemen terakhir tabel \*/  /\* I.S. Tabel tidak kosong \*/  /\* F.S. X adalah nilai elemen terakhir T sebelum penghapusan, \*/  /\* Banyaknya elemen tabel berkurang satu \*/  /\* Tabel T mungkin menjadi kosong \*/  void DelEli (TabPesanan \* T, int i, PesananID \* X);  /\* Menghapus elemen ke-i tabel tanpa mengganggu kontiguitas \*/  /\* I.S. Tabel tidak kosong, i adalah indeks efektif yang valid \*/  /\* F.S. X adalah nilai elemen ke-i T sebelum penghapusan \*/  /\* Banyaknya elemen tabel berkurang satu \*/  /\* Tabel T mungkin menjadi kosong \*/  /\* Proses : Geser elemen ke-i+1 s.d. elemen terakhir \*/  /\* Kurangi elemen efektif tabel \*/  void SubKesabaranArray (TabPesanan\* T, int\* Life);  /\* Menampilkan informasi kesabaran pengunjung menunggu makanan \*/ |

## Binary Tree

Pada ADT Binary Tree (file header : bintree.h, file body : bintree.c), kami menggunakan struktur data pohon biner. Alasan kami memilih binary tree adalah karena pada program simulasi restoran, diperlukan untuk menyimpan dan menampilkan resep dari makanan. Ada beberapa tipe buatan pada ADT Binary Tree, yaitu BinTree. BinTree adalah tipe yang digunakan untuk menyimpan dan menampilkan data resep makanan. Kode programnya dapat dilihat sebagai berikut.

typedef struct tNode \*addrNode;

typedef struct tNode

{

Kata info;

addrNode left;

addrNode right;

} Node;

typedef addrNode BinTree;

Dari kode diatas dapat dilihat bahwa kita membutuhkan ADT lain untuk menyusun tipe buatan seperti tipe Kata dan boolean.

ADT Binary Tree menyelesaikan beberapa permasalahan yaitu menyimpan dan menampilkan data resep makanan yang ada pada restoran.

Beberapa primitif yang kami gunakan pada ADT ini adalah

|  |
| --- |
| BinTree Tree(Kata Akar, BinTree L, BinTree R);  /\* Menghasilkan sebuah pohon biner dari A, L, dan R, jika alokasi berhasil \*/  /\* Menghasilkan pohon kosong (Nil) jika alokasi gagal \*/  void MakeTree(Kata Akar, BinTree L, BinTree R, BinTree \*P);  /\* I.S. Akar, L, R terdefinisi. P Sembarang \*/  /\* F.S. Membentuk pohon P dengan Akar(P)=Akar, Left(P)=L, dan Right(P)=R  jika alokasi berhasil. P = Nil jika alokasi gagal. \*/  BinTree BuildBalanceTree(int n);  /\* Menghasilkan sebuah balanced tree dengan n node, nilai setiap node dibaca \*/  void BuildTree(BinTree \*P, int\* idx);  /\* Menghasilkan sebuah tree dengan idx, nilai setiap idx dibaca \*/  addrNode AlokNode(Kata X);  /\* Mengirimkan addrNode hasil alokasi sebuah elemen \*/  /\* Jika alokasi berhasil, maka addrNode tidak Nil, dan misalnya menghasilkan P, maka Akar(P) = X, Left(P) = Nil, Right(P)=Nil \*/  /\* Jika alokasi gagal, mengirimkan Nil \*/  void DealokNode(addrNode P);  /\* I.S. P terdefinisi \*/  /\* F.S. P dikembalikan ke sistem \*/  /\* Melakukan dealokasi/pengembalian addrNode P \*/  boolean IsTreeEmpty(BinTree P);  /\* Mengirimkan true jika P adalah pohon biner kosong \*/  boolean IsTreeOneElmt(BinTree P);  /\* Mengirimkan true jika P adalah pohon biner tidak kosong dan hanya memiliki 1 elemen \*/  boolean IsUnerLeft(BinTree P);  /\* Mengirimkan true jika pohon biner tidak kosong P adalah pohon unerleft: hanya mempunyai subpohon kiri \*/  boolean IsUnerRight(BinTree P);  /\* Mengirimkan true jika pohon biner tidak kosong P adalah pohon unerright: hanya mempunyai subpohon kanan\*/  boolean IsBiner(BinTree P);  /\* Mengirimkan true jika pohon biner tidak kosong P adalah pohon biner: mempunyai subpohon kiri dan subpohon kanan\*/  void PrintPreorder(BinTree P);  /\* I.S. P terdefinisi \*/  /\* F.S. Semua simpul P sudah dicetak secara preorder: akar, pohon kiri, dan pohon kanan.  Setiap pohon ditandai dengan tanda kurung buka dan kurung tutup ().  Pohon kosong ditandai dengan ().  Tidak ada tambahan karakter apa pun di depan, tengah, atau akhir. \*/  /\* Contoh:  (A()()) adalah pohon dengan 1 elemen dengan akar A  (A(B()())(C()())) adalah pohon dengan akar A dan subpohon kiri (B()()) dan subpohon kanan (C()()) \*/  void PrintInorder(BinTree P);  /\* I.S. P terdefinisi \*/  /\* F.S. Semua simpul P sudah dicetak secara inorder: pohon kiri, akar, dan pohon kanan.  Setiap pohon ditandai dengan tanda kurung buka dan kurung tutup ().  Pohon kosong ditandai dengan ().  Tidak ada tambahan karakter apa pun di depan, tengah, atau akhir. \*/  /\* Contoh:  (()A()) adalah pohon dengan 1 elemen dengan akar A  ((()B())A(()C())) adalah pohon dengan akar A dan subpohon kiri (()B()) dan subpohon kanan (()C()) \*/  void PrintPostorder(BinTree P);  /\* I.S. P terdefinisi \*/  /\* F.S. Semua simpul P sudah dicetak secara postorder: pohon kiri, pohon kanan, dan akar.  Setiap pohon ditandai dengan tanda kurung buka dan kurung tutup ().  Pohon kosong ditandai dengan ().  Tidak ada tambahan karakter apa pun di depan, tengah, atau akhir. \*/  /\* Contoh:  (()()A) adalah pohon dengan 1 elemen dengan akar A  ((()()B)(()()C)A) adalah pohon dengan akar A dan subpohon kiri (()()B) dan subpohon kanan (()()C) \*/  void PrintTreeRek (BinTree P, int h, int CLevel);  void PrintTree(BinTree P, int h);  /\* I.S. P terdefinisi, h adalah jarak indentasi (spasi) \*/  /\* F.S. Semua simpul P sudah ditulis dengan indentasi (spasi) \*/  /\* Penulisan akar selalu pada baris baru (diakhiri newline) \*/  /\* Contoh, jika h = 2:  1) Pohon preorder: (A()()) akan ditulis sbb:  A  2) Pohon preorder: (A(B()())(C()())) akan ditulis sbb:  A  B  C  3) Pohon preorder: (A(B(D()())())(C()(E()()))) akan ditulis sbb:  A  B  D  C  E  \*/  boolean SearchTree(BinTree P, Kata X);  /\* Mengirimkan true jika ada node dari P yang bernilai X \*/  int NbElmtTree(BinTree P);  /\* Mengirimkan banyaknya elemen (node) pohon biner P \*/  int NbDaun(BinTree P);  /\* Mengirimkan banyaknya daun (node) pohon biner P \*/  /\* Prekondisi: P tidak kosong \*/  boolean IsSkewLeft(BinTree P);  /\* Mengirimkan true jika P adalah pohon condong kiri \*/  /\* Pohon kosong adalah pohon condong kiri \*/  boolean IsSkewRight(BinTree P);  /\* Mengirimkan true jika P adalah pohon condong kanan \*/  /\* Pohon kosong adalah pohon condong kanan \*/  int Level(BinTree P, Kata X);  /\* Mengirimkan level dari node X yang merupakan salah satu simpul dari pohon biner P.  Akar(P) level-nya adalah 1. Pohon P tidak kosong. \*/  int Tinggi(BinTree P);  /\* Pohon Biner mungkin kosong. Tinggi pohon kosong = 0.  Mengirim "height" yaitu tinggi dari pohon \*/ |

## Jam

Pada ADT Jam (file header : jam.h, file body : jam.c), kami menggunakan struktur data jam. Alasan kami memilih jam adalah karena pada program simulasi restoran, digunakan untuk menyimpan waktu ketika melakukan ​save. Ada beberapa tipe buatan pada ADT Jam, yaitu Jam. Jam adalah tipe yang digunakan untuk menyimpan waktu ketika melakukan save. Kode programnya dapat dilihat sebagai berikut.

typedef struct {

int HH; /\* integer [0..23] \*/

int MM; /\* integer [0..59] \*/

int SS; /\* integer [0..59] \*/

} JAM;

Dari kode diatas dapat dilihat bahwa kita membutuhkan ADT lain untuk menyusun tipe buatan seperti tipe boolean.

ADT Jam menyelesaikan beberapa permasalahan yaitu untuk menyimpan waktu ketika melakukan save.

Beberapa primitif yang kami gunakan pada ADT ini adalah

|  |
| --- |
| boolean IsJAMValid (int H, int M, int S);  /\* Mengirim true jika H,M,S dapat membentuk J yang valid \*/  /\* dipakai untuk mentest SEBELUM membentuk sebuah Jam \*/  JAM MakeJAM (int HH, int MM, int SS);  /\* Membentuk sebuah JAM dari komponen-komponennya yang valid \*/  /\* Prekondisi : HH, MM, SS valid untuk membentuk JAM \*/  void BacaJAM (JAM \* J);  /\* I.S. : J tidak terdefinisi \*/  /\* F.S. : J terdefinisi dan merupakan jam yang valid \*/  /\* Proses : mengulangi membaca komponen HH, MM, SS sehingga membentuk J \*/  /\* yang valid. Tidak mungkin menghasilkan J yang tidak valid. \*/  /\* Pembacaan dilakukan dengan mengetikkan komponen HH, MM, SS  dalam satu baris, masing-masing dipisahkan 1 spasi, diakhiri enter. \*/  /\* Jika JAM tidak valid maka diberikan pesan: "Jam tidak valid", dan pembacaan  diulangi hingga didapatkan jam yang valid. \*/  /\* Contoh:  60 3 4  Jam tidak valid  1 3 4  --> akan terbentuk JAM <1,3,4> \*/    void TulisJAM (JAM J);  /\* I.S. : J sembarang \*/  /\* F.S. : Nilai J ditulis dg format HH:MM:SS \*/  /\* Proses : menulis nilai setiap komponen J ke layar dalam format HH:MM:SS  tanpa karakter apa pun di depan atau belakangnya, termasuk spasi, enter, dll.\*/  long JAMToDetik (JAM J);  /\* Diberikan sebuah JAM, mengkonversi menjadi jumlah detik dari pukul 0:0:0 \*/  /\* Rumus : detik = 3600\*HH + 60\*MM + SS \*/  /\* Nilai maksimum = 3600\*23+59\*60+59 \*/  JAM DetikToJAM (long N);  /\* Mengirim konversi detik ke JAM \*/  /\* Catatan: Jika N >= 86400, maka harus dikonversi dulu menjadi jumlah detik yang mewakili jumlah detik yang mungkin dalam 1 hari, yaitu dengan rumus:  N1 = N mod 86400, baru N1 dikonversi menjadi JAM \*/  boolean JEQ (JAM J1, JAM J2);  /\* Mengirimkan true jika J1=J2, false jika tidak \*/  boolean JNEQ (JAM J1, JAM J2);  /\* Mengirimkan true jika J1 tidak sama dengan J2 \*/  boolean JLT (JAM J1, JAM J2);  /\* Mengirimkan true jika J1<J2, false jika tidak \*/  boolean JGT (JAM J1, JAM J2);  /\* Mengirimkan true jika J1>J2, false jika tidak \*/  JAM NextDetik (JAM J);  /\* Mengirim 1 detik setelah J dalam bentuk JAM \*/  JAM NextNDetik (JAM J, int N);  /\* Mengirim N detik setelah J dalam bentuk JAM \*/  JAM PrevDetik (JAM J);  /\* Mengirim 1 detik sebelum J dalam bentuk JAM \*/  JAM PrevNDetik (JAM J, int N);  /\* Mengirim N detik sebelum J dalam bentuk JAM \*/  long Durasi (JAM JAw, JAM JAkh);  /\* Mengirim JAkh-JAw dlm Detik, dengan kalkulasi \*/  /\* Jika JAw > JAkh, maka JAkh adalah 1 hari setelah JAw \*/  JAM CurrentTime();  /\* Mengirim real-time ke layar \*/ |

## List Rekursif

Pada ADT List Rekursif (file header : listrek.h, file body : listrek.c), kami menggunakan struktur data list rekursif. Alasan kami memilih list rekursif adalah karena pada program simulasi restoran, digunakan untuk implementasi denah ruangan, implementasi resep, dan implementasi Graf. Ada beberapa tipe buatan pada ADT List Rekursif, yaitu List. List adalah tipe yang digunakan untuk mengimplementasikan denah ruangan, resep, dan Graf. Kode programnya dapat dilihat sebagai berikut.

typedef int infotype;

typedef struct tElmtlist\* address;

typedef struct tElmtlist {

infotype info;

address next;

} ElmtList;

typedef address List;

ADT List Rekursif menyelesaikan beberapa permasalahan yaitu untuk mengimplementasikan denah ruangan pada restoran, untuk mengimplementasikan bahan resep makanan, dan juga mengimplementasikan Graf dengan memanfaatkan ADT List.

Beberapa primitif yang kami gunakan pada ADT ini adalah

|  |
| --- |
| int IsOneElmt(List L);  /\* Mengirimkan 1 jika L berisi 1 elemen dan 0 jika > 1 elemen atau kosong \*/  infotype FirstElmt (List L);  /\* Mengirimkan elemen pertama sebuah list L yang tidak kosong \*/  List Konso(infotype e, List L);  /\* Mengirimkan list L dengan tambahan e sebagai elemen pertamanya.  e dialokasi terlebih dahulu. Jika alokasi gagal, mengirimkan L. \*/  List KonsB(List L, infotype e);  /\* Mengirimkan list L dengan tambahan e sebagai elemen pertamanya \*/  /\* e harus dialokasi terlebih dahulu \*/  /\* Jika alokasi e gagal, mengirimkan L \*/  List Copy (List L);  /\* Mengirimkan salinan list L (menjadi list baru) \*/  /\* Jika ada alokasi gagal, mengirimkan L \*/  void MCopy (List Lin, List \*Lout);  /\* I.S. Lin terdefinisi \*/  /\* F.S. Lout berisi salinan dari Lin \*/  /\* Proses : menyalin Lin ke Lout \*/  List Concat (List L1, List L2);  /\* Mengirimkan salinan hasil konkatenasi list L1 dan L2 (menjadi list baru) \*/  /\* Jika ada alokasi gagal, menghasilkan NULL \*/  void MConcat (List L1, List L2, List \*LHsl);  /\* I.S. L1, L2 terdefinisi \*/  /\* F.S. LHsl adalah hasil melakukan konkatenasi L1 dan L2 dengan cara "disalin" \*/  /\* Proses : Menghasilkan salinan hasil konkatenasi list L1 dan L2 \*/  void PrintList (List L);  /\* I.S. L terdefinisi. \*/  /\* F.S. Setiap elemen list dicetak. \*/  int NbElmtList (List L);  /\* Mengirimkan banyaknya elemen list L, Nol jika L kosong \*/  boolean Search (List L, infotype X);  /\* Mengirim true jika X adalah anggota list, false jika tidak \*/ |

## Matriks

Pada ADT Matriks (file header : matriks.h, file body : matriks.c), kami menggunakan struktur data matriks. Alasan kami memilih matriks adalah karena pada program simulasi restoran, diperlukan untuk digunakan sebagai representasi sebagian ruangan yang ada pada permainan. Ada beberapa tipe buatan pada ADT Matriks, yaitu Matriks. Matriks adalah tipe yang digunakan untuk digunakan sebagai representasi sebagian ruangan yang ada pada permainan, seperti letak meja, kursi, pengunjung, dan player. Kode programnya dapat dilihat sebagai berikut.

typedef struct {

int Mem[BrsMax+1][KolMax+1];

int NBrsEff; /\* banyaknya/ukuran baris yg terdefinisi \*/

int NKolEff; /\* banyaknya/ukuran kolom yg terdefinisi \*/

} MATRIKS;

ADT Matriks menyelesaikan beberapa permasalahan yaitu sebagai representasi sebagian ruangan yang ada pada permainan, seperti letak meja, kursi, pengunjung, dan player dalam program simulasi tersebut dengan memanfaatkan matriks.

Beberapa primitif yang kami gunakan pada ADT ini adalah

|  |
| --- |
| void MakeMATRIKS (int NB, int NK, MATRIKS \* M);  /\* Membentuk sebuah MATRIKS "kosong" yang siap diisi berukuran NB x NK di "ujung kiri" memori \*/  /\* I.S. NB dan NK adalah valid untuk memori matriks yang dibuat \*/  /\* F.S. Matriks M sesuai dengan definisi di atas terbentuk \*/  boolean IsIdxValid (int i, int j);  /\* Mengirimkan true jika i, j adalah int yang valid untuk matriks apa pun \*/  int GetFirstIdxBrs (MATRIKS M);  /\* Mengirimkan int baris terkecil M \*/  int GetFirstIdxKol (MATRIKS M);  /\* Mengirimkan int kolom terkecil M \*/  int GetLastIdxBrs (MATRIKS M);  /\* Mengirimkan int baris terbesar M \*/  int GetLastIdxKol (MATRIKS M);  /\* Mengirimkan int kolom terbesar M \*/  boolean IsIdxEff (MATRIKS M, int i, int j);  /\* Mengirimkan true jika i, j adalah int efektif bagi M \*/  int GetElmtDiagonal (MATRIKS M, int i);  /\* Mengirimkan elemen M(i,i) \*/  void CopyMATRIKS (MATRIKS MIn, MATRIKS \* MHsl);  /\* Melakukan assignment MHsl MIn \*/  void BacaMATRIKS (MATRIKS \* M, int NB, int NK);  /\* I.S. IsIdxValid(NB,NK) \*/  /\* F.S. M terdefinisi nilai elemen efektifnya, berukuran NB x NK \*/  /\* Proses: Melakukan MakeMATRIKS(M,NB,NK) dan mengisi nilai efektifnya \*/  /\* Selanjutnya membaca nilai elemen per baris dan kolom \*/  /\* Contoh: Jika NB = 3 dan NK = 3, maka contoh cara membaca isi matriks :  1 2 3  4 5 6  8 9 10  \*/  void TulisMATRIKS (MATRIKS M);  /\* I.S. M terdefinisi \*/  /\* F.S. Nilai M(i,j) ditulis ke layar per baris per kolom, masing-masing elemen per baris  dipisahkan sebuah spasi \*/  /\* Proses: Menulis nilai setiap elemen M ke layar dengan traversal per baris dan per kolom \*/  /\* Contoh: menulis matriks 3x3 (ingat di akhir tiap baris, tidak ada spasi)  1 2 3  4 5 6  8 9 10  \*/  MATRIKS TambahMATRIKS (MATRIKS M1, MATRIKS M2);  /\* Prekondisi : M1 berukuran sama dengan M2 \*/  /\* Mengirim hasil penjumlahan matriks: M1 + M2 \*/  MATRIKS KurangMATRIKS (MATRIKS M1, MATRIKS M2);  /\* Prekondisi : M berukuran sama dengan M \*/  /\* Mengirim hasil pengurangan matriks: salinan M1 – M2 \*/  MATRIKS KaliMATRIKS (MATRIKS M1, MATRIKS M2);  /\* Prekondisi : Ukuran kolom efektif M1 = ukuran baris efektif M2 \*/  /\* Mengirim hasil perkalian matriks: salinan M1 \* M2 \*/  MATRIKS KaliKons (MATRIKS M, int X);  /\* Mengirim hasil perkalian setiap elemen M dengan X \*/  void PKaliKons (MATRIKS \* M, int K);  /\* I.S. M terdefinisi, K terdefinisi \*/  /\* F.S. Mengalikan setiap elemen M dengan K \*/  boolean EQMatriks (MATRIKS M1, MATRIKS M2);  /\* Mengirimkan true jika M1 = M2, yaitu NBElmt(M1) = NBElmt(M2) dan \*/  /\* untuk setiap i,j yang merupakan int baris dan kolom M1(i,j) = M2(i,j) \*/  /\* Juga merupakan strong EQ karena GetFirstIdxBrs(M1) = GetFirstIdxBrs(M2)  dan GetLastIdxKol(M1) = GetLastIdxKol(M2) \*/  boolean NEQMatriks (MATRIKS M1, MATRIKS M2);  /\* Mengirimkan true jika M1 tidak sama dengan M2 \*/  boolean EQSize (MATRIKS M1, MATRIKS M2);  /\* Mengirimkan true jika ukuran efektif matriks M1 sama dengan ukuran efektif M2 \*/  /\* yaitu GetBrsEff(M1) = GetNBrsEff (M2) dan GetNKolEff (M1) = GetNKolEff (M2) \*/  int NBElmtMatriks (MATRIKS M);  /\* Mengirimkan banyaknya elemen M \*/  boolean IsBujurSangkar (MATRIKS M);  /\* Mengirimkan true jika M adalah matriks dg ukuran baris dan kolom sama \*/  boolean IsSimetri (MATRIKS M);  /\* Mengirimkan true jika M adalah matriks simetri : IsBujurSangkar(M)  dan untuk setiap elemen M, M(i,j)=M(j,i) \*/  boolean IsSatuan (MATRIKS M);  /\* Mengirimkan true jika M adalah matriks satuan: IsBujurSangkar(M) dan  setiap elemen diagonal M bernilai 1 dan elemen yang bukan diagonal bernilai 0 \*/  boolean IsSparse (MATRIKS M);  /\* Mengirimkan true jika M adalah matriks sparse: mariks “jarang” dengan definisi:  hanya maksimal 5% dari memori matriks yang efektif bukan bernilai 0 \*/  MATRIKS Inverse1 (MATRIKS M);  /\* Menghasilkan salinan M dengan setiap elemen "di-invers", yaitu dinegasikan (dikalikan -1) \*/  float Determinan (MATRIKS M);  /\* Prekondisi: IsBujurSangkar(M) \*/  /\* Menghitung nilai determinan M \*/  void PInverse1 (MATRIKS \* M);  /\* I.S. M terdefinisi \*/  /\* F.S. M di-invers, yaitu setiap elemennya dinegasikan (dikalikan -1) \*/  void Transpose (MATRIKS \* M);  /\* I.S. M terdefinisi dan IsBujursangkar(M) \*/  /\* F.S. M "di-transpose", yaitu setiap elemen M(i,j) ditukar nilainya dengan elemen M(j,i) \*/  void MakeEmptyMatriks(MATRIKS \*M);  /\* Membuat matriks kosong \*/ |

## Mesin Karakter & Mesin Kata

Pada ADT Mesin Karakter dan Mesin kata (file header : mesinkar.h & mesinkata.h, file body : mesinkar.c & mesinkata.c), kami menggunakan struktur data character. Alasan kami memilih character adalah karena pada program simulasi restoran, memerlukan input kata untuk menjalankan perintah yang ada pada program. Ada beberapa tipe buatan pada ADT Mesin Kata, yaitu Kata. Kata adalah tipe yang digunakan untuk bisa menginput kata yang terdiri dari beberapa karakter. Kode programnya dapat dilihat sebagai berikut.

typedef struct {

char TabKata[NMax+1];

int Length;

int ID;

} Kata;

ADT Mesin Kata menyelesaikan beberapa permasalahan yaitu untuk bisa menginput tipe kata (*string*) pada program simulasi restoran, seperti membaca ​command dari pengguna, dan - membaca ​state dari permainan yang sudah pernah disimpan. Permasalahan lain yang dapat diselesaikan adalah membaca informasi yang diperlukan dari file eksternal, misal informasi peta.

Beberapa primitif yang kami gunakan pada ADT Karakter adalah

|  |
| --- |
| void START(char\* filename);  /\* Mesin siap dioperasikan. Pita disiapkan untuk dibaca.  Karakter pertama yang ada pada pita posisinya adalah pada jendela.  I.S. : sembarang  F.S. : CC adalah karakter pertama pada pita  Jika CC != MARK maka EOP akan padam (false)  Jika CC = MARK maka EOP akan menyala (true) \*/  void ADV();  /\* Pita dimajukan satu karakter.  I.S. : Karakter pada jendela = CC, CC != MARK  F.S. : CC adalah karakter berikutnya dari CC yang lama,  CC mungkin = MARK  Jika CC = MARK maka EOP akan menyala (true) \*/  void ADVInt(); |

Beberapa primitif yang kami gunakan pada ADT Karakter adalah

|  |
| --- |
| void IgnoreBlank();  /\* Mengabaikan satu atau beberapa BLANK  I.S. : CC sembarang  F.S. : CC ≠ BLANK atau CC = MARK \*/  void STARTKATA(char\* filename);  /\* I.S. : CC sembarang  F.S. : EndKata = true, dan CC = MARK;  atau EndKata = false, CKata adalah kata yang sudah diakuisisi,  CC karakter pertama sesudah karakter terakhir kata \*/  void ADVKATA();  /\* I.S. : CC adalah karakter pertama kata yang akan diakuisisi  F.S. : CKata adalah kata terakhir yang sudah diakuisisi,  CC adalah karakter pertama dari kata berikutnya, mungkin MARK  Jika CC = MARK, EndKata = true.  Proses : Akuisisi kata menggunakan procedure SalinKata \*/  void SalinKata();  /\* Mengakuisisi kata, menyimpan dalam CKata  I.S. : CC adalah karakter pertama dari kata  F.S. : CKata berisi kata yang sudah diakuisisi;  CC = BLANK atau CC = MARK;  CC adalah karakter sesudah karakter terakhir yang diakuisisi.  Jika panjang kata melebihi NMax, maka sisa kata "dipotong" \*/  Kata StringToKata (char\* s);  boolean IsEqKata (Kata a, Kata b);  void OutputKata(Kata K);  Kata InputKata();  void Length(Kata \*kata);  boolean compareKata(Kata kata1, char\* teks);  void printID(Kata kata1); |

## Point

Pada ADT Point (file header : point.h, file body : point.c), kami menggunakan struktur data point. Alasan kami memilih point adalah karena pada program simulasi restoran, diperlukan untuk menaruh posisi objek yang ada pada program. Ada beberapa tipe buatan pada ADT Point, yaitu Point. Point adalah tipe yang digunakan untuk menaruh posisi objek yang ada pada program simulasi. Kode programnya dapat dilihat sebagai berikut.

typedef struct {

int X; /\* absis \*/

int Y; /\* ordinat \*/

} POINT;

ADT Point menyelesaikan beberapa permasalahan yaitu posisi letaknya objek yang ada pada program, seperti meja, kursi, pengunjung, dan player. ADT Point juga digunakan banyak dalam implementasi yang ada pada program simulasi restoran ini, baik itu dengan bantuan ADT Matriks dan juga ADT lainnya yang ada pada program ini.

Beberapa primitif yang kami gunakan pada ADT ini adalah

|  |
| --- |
| POINT MakePOINT (int X, int Y);  /\* Membentuk sebuah POINT dari komponen-komponennya \*/  void BacaPOINT (POINT \* P);  /\* Membaca nilai absis dan ordinat dari keyboard dan membentuk  POINT P berdasarkan dari nilai absis dan ordinat tersebut \*/  /\* Komponen X dan Y dibaca dalam 1 baris, dipisahkan 1 buah spasi \*/  /\* Contoh: 1 2  akan membentuk POINT <1,2> \*/  /\* I.S. Sembarang \*/  /\* F.S. P terdefinisi \*/  void TulisPOINT (POINT P);  /\* Nilai P ditulis ke layar dengan format "(X,Y)"  tanpa spasi, enter, atau karakter lain di depan, belakang,  atau di antaranya  Output X dan Y harus dituliskan dalam bilangan riil dengan 2 angka di belakang koma.  \*/  /\* I.S. P terdefinisi \*/  /\* F.S. P tertulis di layar dengan format "(X,Y)" \*/  boolean EQ (POINT P1, POINT P2);  /\* Mengirimkan true jika P1 = P2 : absis dan ordinatnya sama \*/  boolean NEQ (POINT P1, POINT P2);  /\* Mengirimkan true jika P1 tidak sama dengan P2 \*/  boolean IsOrigin (POINT P);  /\* Menghasilkan true jika P adalah titik origin \*/  boolean IsOnSbX (POINT P);  /\* Menghasilkan true jika P terletak Pada sumbu X \*/  boolean IsOnSbY (POINT P);  /\* Menghasilkan true jika P terletak pada sumbu Y \*/  int Kuadran (POINT P);  /\* Menghasilkan kuadran dari P: 1, 2, 3, atau 4 \*/  /\* Prekondisi : P bukan titik origin, \*/  /\* dan P tidak terletak di salah satu sumbu \*/  boolean AdjPOINT (POINT a, POINT b); |

## Prioqueue

Pada ADT Prioqueue (file header : prioqueue.h, file body : prioqueue.c), kami menggunakan struktur data queue. Alasan kami memilih queue adalah karena pada program simulasi restoran, diperlukan untuk representasi antrean pengunjung restoran, pesanan, serta kesabaran para pengunjung menunggu pesanan mereka. Ada beberapa tipe buatan pada ADT Prioqueue, yaitu Queue. Queue adalah tipe yang digunakan untuk membuat antrean pada program simulasi restoran, baik itu antrean pengunjung maupun pesanan, serta info kesabaran para pengunjung dalam menunggu pesanan mereka. Kode programnya dapat dilihat sebagai berikut.

typedef struct { int prio; /\* [1..3], prioritas dengan nilai 1..3 (3 adalah prioritas tertinggi) \*/

int info; /\* nilai elemen \*/

int kesabaran ;

} infotypeQueue;

typedef struct { infotypeQueue \* TI; /\* tabel penyimpan elemen \*/

int HEAD; /\* alamat penghapusan \*/

int TAIL; /\* alamat penambahan \*/

int MaxElm; /\* Max elemen queue \*/

} Queue;

ADT Prioqueue menyelesaikan beberapa permasalahan yaitu mensimulasikan antrean para pengunjung untuk memesan makanan dan antrean pesanan para pengunjung dalam program simulasi restoran tersebut. Permasalahan lain yang dapat diselesaikan adalah menunjukkan info kesabaran para pengunjung menunggu pesanan mereka tiba atau disiapkan.

Beberapa primitif yang kami gunakan pada ADT ini adalah

|  |
| --- |
| boolean IsEmptyQueue (Queue Q);  /\* Mengirim true jika Q kosong: lihat definisi di atas \*/  boolean IsFullQueue (Queue Q);  /\* Mengirim true jika tabel penampung elemen Q sudah penuh \*/  /\* yaitu mengandung elemen sebanyak MaxEl \*/  int NBElmtQueue (Queue Q);  /\* Mengirimkan banyaknya elemen queue. Mengirimkan 0 jika Q kosong. \*/  void CreateEmptyQueue (Queue \* Q, int Max);  /\* I.S. sembarang \*/  /\* F.S. Sebuah Q kosong terbentuk dan salah satu kondisi sbb: \*/  /\* Jika alokasi berhasil, Tabel memori dialokasi berukuran Max+1 \*/  /\* atau : jika alokasi gagal, Q kosong dg MaxEl=0 \*/  /\* Proses : Melakukan alokasi, membuat sebuah Q kosong \*/  void DeAlokasiQueue (Queue \* Q);  /\* Proses: Mengembalikan memori Q \*/  /\* I.S. Q pernah dialokasi \*/  /\* F.S. Q menjadi tidak terdefinisi lagi, MaxEl(Q) diset 0 \*/  void AddQueue (Queue \* Q, infotypeQueue X);  /\* Proses: Menambahkan X pada Q dengan aturan priority queue, terurut mengecil berdasarkan prio \*/  /\* I.S. Q mungkin kosong, tabel penampung elemen Q TIDAK penuh \*/  /\* F.S. X menjadi TAIL yang baru, TAIL "maju" dengan mekanisme circular buffer;  elemen baru disisipkan pada posisi yang tepat sesuai dengan prioritas \*/  void DelQueue (Queue \* Q, infotypeQueue \* X);  /\* Proses: Menghapus X pada Q dengan aturan FIFO \*/  /\* I.S. Q tidak mungkin kosong \*/  /\* F.S. X = nilai elemen HEAD pd I.S., HEAD "maju" dengan mekanisme circular buffer;  Q mungkin kosong \*/  void SubKesabaranQueue (Queue\* Q, int\* Life) ;  void PrintQueue (Queue Q);  /\* Mencetak isi queue Q ke layar \*/  /\* I.S. Q terdefinisi, mungkin kosong \*/  /\* F.S. Q tercetak ke layar dengan format:  <prio-1> <elemen-1>  ...  <prio-n> <elemen-n>  #  \*/ |

## Stack

Pada ADT Stack (file header : stackt.h, file body : stackt.c), kami menggunakan struktur data *stack*. Alasan kami memilih stack adalah karena pada program simulasi restoran, diperlukan untuk representasi tumpukan makanan di tangan dan di nampan. Ada beberapa tipe buatan pada ADT Stack, yaitu Stack. Stack adalah tipe yang digunakan untuk menaruh tumpukan makanan di tangan dan di nampan. Kode programnya dapat dilihat sebagai berikut.

typedef struct {

Kata T[MaxEl+1]; /\* tabel penyimpan elemen \*/

int TOP; /\* alamat TOP: elemen puncak \*/

} Stack;

Dari kode diatas dapat dilihat bahwa kita membutuhkan ADT lain untuk menyusun tipe buatan seperti tipe Kata.

ADT Stack menyelesaikan beberapa permasalahan yaitu menaruh dan menampilkan tumpukan makanan di tangan dan di nampan, dengan memanfaatkan ADT Mesin Kata untuk menuliskan nama daftar makanan yang ada pada tumpukan tangan dan nampan.

Beberapa primitif yang kami gunakan pada ADT ini adalah

|  |
| --- |
| void CreateEmptyStack (Stack \*S);  /\* I.S. sembarang; \*/  /\* F.S. Membuat sebuah stack S yang kosong berkapasitas MaxEl \*/  /\* jadi indeksnya antara 1.. MaxEl+1 karena 0 tidak dipakai \*/  /\* Ciri stack kosong : TOP bernilai Nil \*/  boolean IsEmptyStack (Stack S);  /\* Mengirim true jika Stack kosong: lihat definisi di atas \*/  boolean IsFullStack (Stack S);  /\* Mengirim true jika tabel penampung nilai elemen stack penuh \*/  void PushStack (Stack \* S, Kata X);  /\* Menambahkan X sebagai elemen Stack S. \*/  /\* I.S. S mungkin kosong, tabel penampung elemen stack TIDAK penuh \*/  /\* F.S. X menjadi TOP yang baru,TOP bertambah 1 \*/  void PopStack (Stack \* S, Kata\* X);  /\* Menghapus X dari Stack S. \*/  /\* I.S. S tidak mungkin kosong \*/  /\* F.S. X adalah nilai elemen TOP yang lama, TOP berkurang 1 \*/  void TulisStack(Stack S);  /\* Menuliskan semua elemen pada Stack \*/ |

## Graph

Pada ADT Graph (file header : graph.h, file body : graph.c), kami menggunakan struktur data adjacency matrix. Alasan kami memilih adjacency matrix adalah karena pada program simulasi restoran ini, akan memudahkan untuk menghubunkan ruangan satu dengan ruangan yang lainnya pada restoran. Ada beberapa tipe buatan pada ADT Graph, yaitu Graph. Graph adalah tipe yang digunakan untuk menghubungkan masing-masing bagian dari peta agar terbentuk suatu peta besar. Kode programnya dapat dilihat sebagai berikut.

typedef struct {

int Adj[NodeMax+1][NodeMax+1];

int Node; /\* banyaknya node yg terdefinisi \*/

} Graph;

ADT Graph menyelesaikan beberapa permasalahan yaitu digunakan untuk menghubungkan masing-masing bagian dari peta agar terbentuk suatu peta besar dengan memanfaatkan adjacency matrix.

Beberapa primitif yang kami gunakan pada ADT ini adalah

|  |
| --- |
| void MakeGraph (int N, Graph \* G);  /\* Membentuk sebuah MATRIKS "kosong" yang siap diisi berukuran N x N di "ujung kiri" memori \*/  int SearchEdge (Graph G, int A, int B);  /\* Mengecek jenis koneksi antara A dan B \*/  int SearchEdge2 (Graph G, int A, int C);  /\* Mencari node yang terhubung dengan A dengan koneksi C \*/  void CreateEdge (Graph\* G, int A, int B, int C);  /\* Membuat node A dan B terhubung dengan kode C \*/ |

## ADT

Pada ADT bernama ADT (file header : adt.h, file body : adt.c), kami menggunakan struktur data yang terdiri dari semua ADT yang dijelaskan di atas. Kami membuat ADT ini untuk membuat tampilan program simulasi dan objek-objeknya, serta fungsi generate lainnya. Ada beberapa tipe buatan pada ADT, yaitu Player. Player adalah tipe yang digunakan untuk objek player dalam program simulasi tersebut. Kode programnya dapat dilihat sebagai berikut.

typedef struct{

Kata nama;

int money;

int life;

int time;

POINT posisi;

int ruangan;

} Player;

Dari kode diatas dapat dilihat bahwa kita membutuhkan ADT lain untuk menyusun tipe buatan seperti tipe Kata dan POINT.

Selanjutnya adalah MejaMakan, yaitu tipe yang digunakan untuk objek meja makan. Kode programnya dapat dilihat sebagai berikut

typedef struct{

POINT posisi; //mejanya ada di titik berapa

int kursi; //ada berapa customer yang bisa ditempatkan disitu

int NCustomer; //lagi ada berapa orang yang ngedudukin

boolean isOrderTaken;

POINT TChair[5];

} MejaMakan;

Setelah itu, MejaDapur, yaitu tipe yang digunakan untuk objek meja dapur. Kode programnya dapat dilihat sebagai berikut.

typedef struct{

POINT posisi;

Kata bahan;

} MejaDapur;

Ruang, adalah tipe yang digunakan untuk mengimplementasikan ruangan pada restoran. Kode programnya dapat dilihat sebagai berikut.

typedef struct{

POINT P1;

POINT P2;

MejaMakan TTable[5];

}Ruang;

Tray, adalah tipe yang digunakan untuk mengimplementasikan objek tray (nampan). Kode programnya dapat dilihat sebagai berikut.

typedef struct{

POINT posisi;

Stack bahan;

}Tray;

Yang terakhir, RuangDapur, adalah tipe yang digunakan untuk mengimplementasikan ruang dapur yang ada dalam restoran. Kode programnya dapat dilihat sebagai berikut.

typedef struct{

POINT P1;

POINT P2;

Tray T;

MejaDapur M[17];

} RuangDapur;

ADT menyelesaikan beberapa permasalahan yaitu digunakan untuk mengimplementasikan objek yang ada pada program simulasi tersebut, serta men-generate objek-objek pada program secara random. Selain itu juga ADT dapat menyelesaikan permasalahan save-load dalam program.

Beberapa primitif yang kami gunakan pada ADT ini adalah

|  |
| --- |
| void ExitMessage(boolean SaveGame);  void TampilanProgramUtama();  //ntar bakal nampilin program utamanya  void SaveFile();  //bakal ngesave ke file eksternal dgn format yg di tampilan  void LoadFile();  //bakal dipake sama main menu yg Load game  char MatriksToChar(int y);  //ubah isi matriks jadi char  BinTree assignResep();  void init\_ruangan1();  void init\_ruangan2();  void init\_ruangan3();  void init\_dapur();  void InitDenah();  void init\_matriks();  void init\_player();  void init\_all();  void assignMatriks();  int GetRandomMenu();  int GetRandomPrio();  int GetRandomNCust();  int GetRandomArrival();  Kata GetRandomMenuName(int MenuIdx);  void RandomizerQueue(int\* Interval);  Kata SimplMenu (Kata K); |

## Coba Tubes

Pada ADT Coba Tubes (file header : cobatubes.h, file body : cobatubes.c), kami menggunakan struktur data ADT yang bernama ADT yang telah kami jelaskan di atas. Kami membuat ADT Coba Tubes ini untuk menjalan command yang digunakan untuk memberi perintah kepada player dalam program simulasi restoran tersebut.

ADT Coba Tubes menyelesaikan beberapa permasalahan yaitu digunakan untuk menjalankan program seperti memberi command pada program sehingga dapat berjalannya simulasi pada program tersebut. Selain itu juga ADT Coba Tubes dapat menyelesaikan permasalahan berpindah ke ruangan yang lain.

Beberapa primitif yang kami gunakan pada ADT ini adalah

|  |
| --- |
| void GU();  //pemain adalah lokasi dari pemain saat itu,Time adalah waktu yag telah lewat  //Command ini adalah singkatan dari �Go Up�, sehingga posisi player berpindah ke  //atas  void GD();  /\*Command ini adalah singkatan dari �Go Down�, sehingga posisi player berpindah  ke bawah\*/  void GL();  /\*Command ini adalah singkatan dari �Go Left�, sehingga posisi player berpindah  ke kiri\*/  void GR();  /\*Command ini adalah singkatan dari �Go Right�, sehingga posisi player berpindah  kekanan.\*/  void ORDER();//pemain bisa ambil brp banyak order?  /\*Command ini digunakan untuk mengambil order dari meja yang bersebelahan  dengan pemain\*/  void PUT();  /\*Command ini digunakan untuk menaruh makanan di hand ke nampan\*/  void TAKE();  /\*Command ini digunakan untuk mengambil bahan yang bersebelahan dengan  pemain\*/  void CH();  /\*Command ini digunakan untuk membuang seluruh bahan makanan yang  terdapat di tangan pemain\*/  void CT();  /\*Command ini digunakan untuk membuang seluruh makanan yang berada di  dalam tray\*/  void PLACE();  /\*Command ini digunakan untuk menaruh pelanggan di meja dan kosong.  Pelanggan yang ditaruh adalah pelanggan pada top of queue\*/  void GIVE();  /\*Memberikan makanan yang berada di paling atas tumpukan ke pengunjung yang  bertetanggaan\*/  void RECIPE();  /\*Command ini digunakan untuk menampilkan pohon makanan.\*/  void SAVE();  /\*Command ini digunakan untuk menyimpan state permainan saat ini agar dapat  dilanjutkan kemudian. Metode interaksi bebas\*/  void LOAD();  /\*Command ini digunakan untuk melanjutkan permainan sesuai dengan state yang  ada pada text file. Metode interaksi bebas\*/  void EXIT(boolean \*exit);  /\*Command ini digunakan untuk keluar dari program.\*/  int IsNearTable ();  void GantiRuangan ();  void UpdateTimePatience();  Kata IsNearKitchenTable();  void SubKesabaranArray (TabPesanan\* T, int\* Life);  void help(); |

# Program Utama

Program utama ini terdiri dari Initialize dan Play. Initialize berfungsi untuk memunculkan intro dari program simulasi ini, kemudian meminta user untuk memulai game baru atau melanjutkan game. Jika user memilih game baru maka akan dibuat data baru, sedangkan jika memilih melanjutkan game maka akan memakai save data yang sudah disimpan di file eksternal. Kemudian menjalankan Play yang berfungsi untuk menjalankan permainan, fungsi ini merupakan fungsi looping dimana user akan memasukkan input command, seperti “GU”, “TAKE”, “PUT”, “ORDER”, dan lainya, untuk menggerakkan program simulasi ini sampai loopnya berhenti dimana user exit dari program.

# Algoritma-Algoritma Menarik

## Pindah Ruangan

Salah satu algoritma yang kami kira menarik dalam program kami adalah di dalam fungsi pergerakan : GU, GL, GD, GR, ada bagian dimana program mengecek posisi dan arah gerak pemain untuk menentukan apakah di berpindah ruangan. Apabila pemain berpindah ruangan, program memanggil fungsi gantiRuangan() dimana dicek dari ruang dan pintu manakah pemain berpindah. Lalu fungsi tersebut mencari ruangan mana pemain akan berpindah dengan melakukan linear search pada graph dengan ruangan sebagai nodenya. Kami merasa algoritma ini menarik karena cukup rumit tapi terpaksa karena dalam spek harus menggunakan graf.

# Data Test

Isi dengan data test dan penjelasan yang diperlukan. Hal-hal yang harus dijelaskan untuk tiap data test:

* Fitur apa (saja) yang dites dengan data test ini
* Hasil apa yang seharusnya diberikan
* Penjelasan lain (jika ada)
* Data test-nya sendiri.

## <Data Test 1>

Isi dengan penjelasan mengenai data test.

## <Data Test 2>

Isi dengan penjelasan mengenai data test.

# Test Script

Isi dengan skenario test yang dimungkinkan untuk semua fitur yang ada. Bisa dibuat dalam bentuk tabel sebagai berikut:

| **No.** | **Fitur yang Dites** | **Tujuan Testing** | **Langkah-Langkah Testing** | **Input Data Test** | **Hasil yang Diharapkan** | **Hasil yang Keluar** |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | <Fitur 1> | … | … | <tinggal refer ke data test di atas> | … |  |
| 2 | <Fitur 1> | … | … | … | … |  |
| 3 |  |  |  |  |  |  |
| … |  |  |  |  |  |  |
| 5 | <Fitur 2> | … | … | … | … |  |
| 6 | <Fitur 2> | … | … | … | … |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

# Pembagian Kerja dalam Kelompok

|  |  |  |
| --- | --- | --- |
| **NIM** | **Nama** | **Tugas** |
| 13517004 | Bimo Adityarahman W. | Membuat ADT Graph |
| 13517025 | Ricky Yuliawan | Membuat ADT Prioqueue, ADT Mesin Kata, ADT Binary Tree, Membuat Implementasi Resep |
| 13517058 | Ahmad Rizqee Nurhani |  |
| 13517064 | Naufal Aditya D. |  |
| 13517076 | Ahmad Rizal Alifio |  |

# Lampiran

## Deskripsi Tugas Besar

Engi’s Kitchen adalah restoran yang dirintis tahun lalu oleh Chef. Hanya dalam jangka waktu satu tahun, Engi’s Kitchen sudah tergolong sangat sukses dan berjaya. Oleh karena itu, Chef sebagai seorang ​engineer selalu ingin berkembang dan berusaha untuk mengembangkan bisnisnya lebih lanjut.

Pada tahun ini, Chef berani mengembangkan restorannya menjadi lebih besar. Sekarang pengunjung dapat menikmati makanannya di dalam restoran. Karena Chef ingin menjaga kualitas restorannya, makanan yang disajikan harus segar sehingga pembuatan makan harus dilakukan ​on-the-spot. Sebelum ia mengambil resiko untuk merenovasi dan mengembangkan bisnisnya, Chef meminta Anda untuk membuat sebuah program simulasi berjalannya restoran ini.

Restoran Engi’s Kitchen memiliki 3 ruang yang digunakan untuk tamu dan 1 ruang yang digunakan sebagai dapur. Ruangan-ruangan ini terhubung dengan pintu. Pada setiap ruangan terdapat meja-meja yang disusun sedemikian rupa. Meja tersebut beragam kapasitasnya. Ada meja yang hanya bisa untuk berdua, dan ada juga yang mampu menampung 4 pengunjung.

Pengunjung yang mendatangi restoran akan mengantre terlebih dahulu. Interval waktu kedatangan pelanggan dibebaskan. Chef lalu akan mengarahkan pengunjung yang berada pada antrean paling depan untuk duduk di meja yang cukup kapasitasnya. Setelah pengunjung duduk di meja, mereka akan langsung melakukan pemesanan makanan. Pengunjung yang sedang mengantre diharapkan menunggu maksimal selama 30 satuan waktu, jika pengunjung tersebut menunggu lebih lama maka pengunjung akan pulang dan kredibilitas restoran berkurang.

Meskipun terdapat beberapa varietas makanan yang dijual di Engi’s Kitchen. Dipastikan bahwa 1 meja hanya memesan 1 jenis makanan. Pesanan makanan tersebut akan disimpan pada suatu daftar, dan setiap pengunjung yang duduk juga memiliki batas waktu kesabaran seperti pada antrean. Namun, batas waktu kesabaran ini bervariasi setiap pengunjung.

Pembuatan makanan dilakukan dengan menambahkan bahan makanan kepada piring. Pencampuran bahan makanan harus mengikuti urutan tertentu berdasarkan pohon makanan. Makanan yang telah terbentuk harus diletakkan di nampan terlebih dahulu sebelum dibawa ke pengunjung. Nampan mampu menampung 5 tumpuk makanan untuk sekali jalan. Makanan pada nampan berbentuk tumpukan.

Setiap aksi yang dilakukan oleh Chef memakan 1 tick waktu, pada setiap tick mungkin terjadi penambahan pengunjung baru. Setiap 1 tick, kesabaran pengunjung berkurang 1 satuan. Chef sangat berharap dengan adanya program ini ia dapat mengatur strategi agar Engi’s Kitchen semakin sukses dan terus berkembang kedepannya.

## Notulen Rapat

* Senin, 29 Oktober 2018

Kelompok kami mengadakan kumpul perdana di selasar ding dong Labtek V membahas pembagian tugas yang akan dikerjakan untuk Tugas Besar ini.

* Kamis, 1 November 2018

Kelompok kami melakukan asistensi dengan asisten di Lab. Programming, untuk asistensi pertama ini kami lebih banyak membahas rancangan program yang kami buat, serta bertanya tentang spesifikasi yang ada pada Tugas Besar.

* Selasa, 13 November 2018

Kelompok kami melakukan asistensi yang kedua dengan asisten di Lab. Programming. Dalam asistensi yang kedua ini, kami sudah membuat sebagian program dan beberapa ADT yang diperlukan untuk program tugas besar ini. Kami juga bertanya beberapa spesifikasi tugas besar yang menurut kami masih kurang paham, seperti pembuatan ADT Graph dan soal pemakaian real-time di ADT Jam.

* Jum’at, 23 November 2018

Kelompok kami melakukakn asistensi yang ketiga dengan asisten di Lab. Programming. Dalam asistensi yang ketiga ini, kami lebih membahas detail yang perlu diperbaiki lagi dari program yang telah kami buat.

* Di antara rentang waktu-waktu di atas, kami melakukan tugas kami masing-masing dengan mengupload di github dan juga biasanya bertanya progress kami sudah sejauh mana di kelas.
* Jum’at, 23 November 2018 – Sabtu, 24 November 2018

Kelompok kami melakukan *finishing* terhadap program yang kami buat sampai sebagus mungkin, melakukan debug terhadap program yang kami buat, dan juga kami sekaligus membuat laporan tugas besar.

## Log Activity Anggota Kelompok

## Form Asistensi