LAPORAN TUGAS AKHIR
ALGORITMA DAN STRUKTUR DATA

EXSEND

Laporan ini disusun oleh :

Lemuel Lancaster	/ 00000027690
Andrio Effendi	/ 00000026963
Jennie Florensia	/ 00000027184
Josuan Leonardo	/ 00000027571

Program Studi Informatika

Fakultas Teknik dan Informatika

Universitas Multimedia Nusantara

Mei 2019

BAB I
PENDAHULUAN

1.1 Latar Belakang

Perkembangan ilmu pengetahuan dan teknologi informasi semakin pesat sehingga sebagian besar aktivitas manusia bergantung dari penggunaan perangkat teknologi informasi. Salah satu contoh pemanfaatan teknologi informasi adalah dalam bidang jasa pengiriman yaitu dengan penggunaan aplikasi yang dapat memudahkan proses pengiriman barang dari penjual ke pihak penyedia jasa pengiriman sampai ke tangan konsumen. Di era digital ini, jasa pengiriman sangatlah diperlukan khususnya untuk menjadikan waktu kerja semakin efektif, apalagi dengan maraknya *online shop* yang muncul. Banyak perusahaan-perusahaan atau *start-up* baru yang juga memerlukan jasa pengiriman yang dapat diandalkan untuk mendukung pertumbuhan usahanya. Oleh karena itu untuk menyelesaikan masalah ini kami membuat sebuah aplikasi yang bernama EXSend.

Aplikasi EXSend ini dikembangkan dengan menggunakan bahasa pemrograman yang berbasis C yang menggunakan algoritma Dijkstra untuk mencari jalur tercepat dari lokasi konsumen ke lokasi tujuan. Sesuai dengan instuksi yang diberikan oleh Dosen Pengajar, kami menggunakan CodeBlock sebagai perangkat lunak utama dalam pengembangan aplikasi ini dan berberapa aplikasi lainnya untuk memudahkan pekerjaan kami. Pembuatan aplikasi ini melibatkan seluruh anggota kelompok.

Dengan adanya aplikasi EXSend ini, maka diharapkan dapat membantu pengguna jasa pengiriman dan penyedia jasa pengiriman barang untuk dapat melakukan transaksi pengiriman dengan mudah, cepat, dan aman.

1.2 Perumusan Masalah

1. Bagimana membuat jasa pengiriman dengan memanfaatkan perkembangan teknologi saat ini?
2. Bagaimana mempermudah, mempercepat, dan mempercepat proses pengiriman bagi usaha baru?

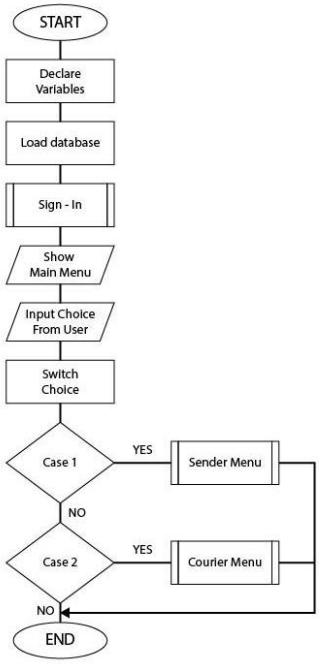
1.3 Tujuan

Adapun tujuan dari pembuatan aplikasi ini adalah untuk mempermudah proses pengiriman barang dari penjual ke pembeli secara cepat dan aman serta dapat mendorong pertumbuhan bisnisnya. Kondisi masyarakat di era digital ini memiliki kehidupan sangat sibuk dan ingin segala sesuatu serba cepat, mudah, dan efisien.

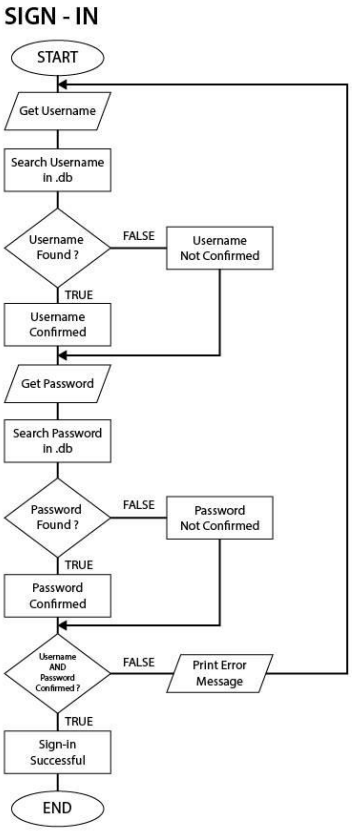
BAB II

PERANCANGAN

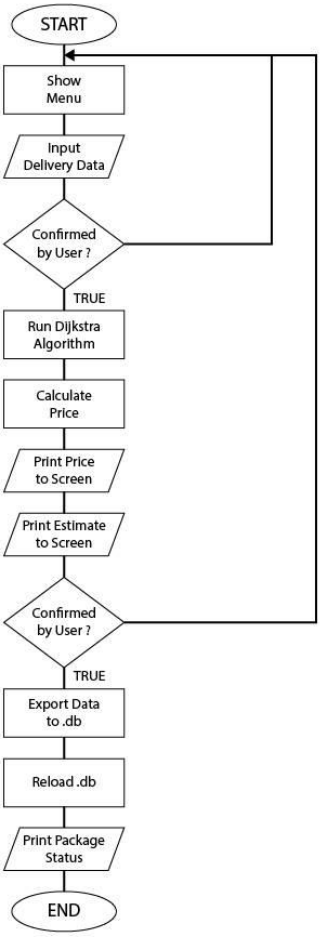
2.2 Penjelasan Diagram Alir



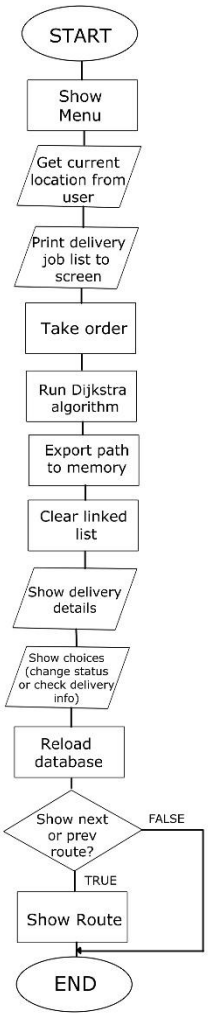
Flowchart keseluruhan sistem program. Pada saat program jalan, pengguna diminta untuk *sign-in* terlebih dahulu. Setelah *sign-in* akan muncul menu utama. Pada menu utama terdapat dua pilihan bagi pengguna, yakni sebagai *sender* (pengirim) dan sebagai *courier* (kurir).



Pada menu *sign-in* ini, pengguna harus mengisi *username* dan *password*. Jika *username* dan *password* yang dimasukkan oleh pengguna benar, maka *sign-in* berhasil dan akan muncul menu utama aplikasi ini.



Pada menu *sender*, pengguna diminta untuk memasukkan data berupa nama pengirim, kota asal pengirim, lokasi pengambilan barang, nama penerima, kota asal penerima, dan alamat penerima. Selanjutnya, program akan menjalankan algoritma Dijkstra untuk mengkalkulasikan rute terpendek dari kota pengirim ke kota penerima, harga jasa pengiriman, serta perkiraan waktu paket sampai.



Pada menu *courier*, pengguna akan diminta untuk memasukkan lokasi saat ini dan program akan menampilkan data pengiriman. Jika pengguna memilih untuk menerima pekerjaan pengiriman yang ditampilkan, maka program akan menjalankan algoritma Dijkstra untuk melakukan kalkulasi jarak terdekat dari lokasi terkini pengguna ke tempat pengambilan paket. Pengguna diberikan pilihan untuk melakukan pembaruan status pengiriman (*Ongoing/Cancelled/Delivered*) dan juga pilihan untuk melihat rute perjalanan selanjutnya atau sebelumnya.

BAB III

IMPLEMENTASI HASIL

3.1 Potongan *Source Code*

```
16  // Struct Declare
17  struct route {
18      char location_route[1][999];
19      struct route *next, *prev;
20  };
```

Pendeklarasian *struct* yang menampung data untuk setiap titik pada graf.

```
871 void search(char *search)
872 {
873     // Declare variable
874     int mid, low, high;
875     low = 0;
876     int counter = 0;
877
878     // Set high variable according to global id
879     if(global_id == 1)
880     {
881         high = user_count - 1;
882     }
883
884     if(global_id == 2)
885     {
886         high = user_count - 1;
887     }
888
889     if(global_id == 51)
890     {
891         high = location_count - 1;
892     }
```

```
894     /// Sequential Search
895     switch(global_id)
896     {
897         case 1 :
898         {
899             for(counter = 0; counter < user_count; counter++)
900             {
901                 if (strcmpi(search, user[counter]) == 0)
902                 {
903                     login_check = 1;
904                 }
905             }
906             break;
907         }
908
909         case 2 :
910         {
911             for(counter = 0; counter < user_count; counter++)
912             {
913                 if (strcmpi(search, password[counter]) == 0)
914                 {
915                     login_check = 1;
916                     user_privelege = privilege[counter];
917                 }
918             }
919             break;
920         }
```

```
922         case 51 :
923         {
924             for(counter = 0; counter < location_count; counter++)
925             {
926                 if (strcmpi(search, location[counter]) == 0)
927                 {
928                     if(sord == 0)
929                     {
930                         source = counter;
931                     }
932                     if(sord == 1)
933                     {
934                         destination = counter;
935                     }
936                 }
937             }
938             break;
939         }
940
941         case 42 :
942         {
943             for(counter = 0; counter < order_count; counter++)
944             {
945                 if (strcmpi(search, sender_name[counter]) == 0)
946                 {
947                     user_order_id[1][user_order_count] = counter;
948                     user_order_count++;
949                     printf("\t\t\t\t\t"); printf("%d) %s | %s | %s", user_order_count, order_stat
950                 }
951             }
952             break;
953         }
954     }
955 }
```

Sequential search digunakan untuk mencari data pengguna saat *sign-in* seperti yang telah dijelaskan pada diagram alir sebelumnya. Pengguna akan diminta melakukan *input* berupa *username* dan *password*, setelah itu akan dilakukan pencarian di dalam *database*, jika terdaftar maka pengguna dapat melakukan *sign-in*. Penggunaan fitur *sign-in* dilakukan agar dapat menjaga privasi pengguna.

```
973 /// Dijkstra Algorithm
974 int minDistance(int dist[],bool sptSet[])
975 {
976
977     // Initialize min value
978     int min = INT_MAX, min_index;
979
980     for (int v = 0; v < node_count; v++)
981         if (sptSet[v] == false && dist[v] <= min)
982             min = dist[v], min_index = v;
983
984     return min_index;
985 }
```

```
1031    void dijkstra()
1032    {
1033        // store shortest distance
1034        int dist[node_count];
1035
1036        // Declare sptSet
1037        bool sptSet[node_count];
1038
1039        // Store shortest path tree
1040        int parent[node_count];
1041
1042        // Initialize all distances as infinite and set sptSet to false
1043        for (int i = 0; i < node_count; i++)
1044        {
1045            parent[source] = -1;
1046            dist[i] = INT_MAX;
1047            sptSet[i] = false;
1048        }
1049
1050        // Set distance from source to source is 0
1051        dist[source] = 0;
```

```
1053        // Find shortest path from all vertex
1054        for (int count = 0; count < node_count - 1; count++)
1055        {
1056            // Choose minimum distance
1057            int u = minDistance(dist, sptSet);
1058
1059            // Set the chosen vertex to true
1060            sptSet[u] = true;
1061
1062            // Update distance value
1063            for (int v = 0; v < node_count; v++)
1064
1065                // Update if not in sptSet
1066                if (!sptSet[v] && graph[u][v] &&
1067                    dist[u] + graph[u][v] < dist[v])
1068                {
1069                    parent[v] = u;
1070                    dist[v] = dist[u] + graph[u][v];
1071                }
1072        }
1073
1074        // Export result to memory
1075        export_dijkstra(dist, node_count, parent);
1076    }
```

Pengimplementasian algoritma Djikstra. Algoritma ini digunakan untuk mencari jalur/rute terpendek dari dua lokasi (*input* oleh pengguna). Adanya pemanggilan fungsi “export_dijkstra” dilakukan untuk mencatat rute yang dilalui agar bisa ditampilkan kepada pengguna saat penggunaan aplikasi.

3.2 *Pseudocode*

Berikut adalah potongan *pseudocode* yang merujuk kepada cara kerja program secara keseluruhan :

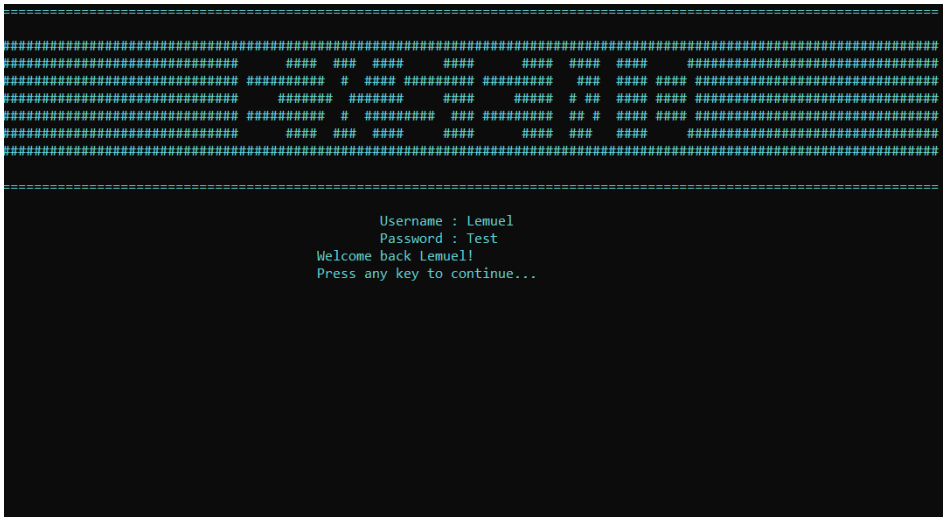
```
Program Start
|
Load database file
        |       Import City Name and IDs to Memory
        |       Import Graph to Memory
        |       Import Delivery Database to Memory
        |       Import User Database
        |       Validate database
Show Menu
        Sign Up
        |       Demand User Information for Registration
        |       Export to User Database
        |       Reload Database
        Sign In
        |       Demand Login Information
        |       IF Yes; Continue;
        While Sender
                |       Show Menu
                |       Demand User Input
                Sender Name
                        |       Insert Name
                Receiver Name
                        |       Insert Name
                Sender Address
                        |       Insert Sender Address
                Pickup City Location
                        |       Show all available location
                        |       Check if Pickup location same as Receiver City → Back to City Location


    Receiver Address
            |       Insert Receiver Address
    Receiver City Location
            |       Show all available location
            |       Check if Receiver Location same as Pickup Location → Back to City Location
    | IF user_input = 6 continue;
    Run Dijkstra Algorithm from pickup location
            |       Calculate Price per km
            |       Show Price
            |       Show Est. Time
    User OK!
            |       Export to Delivery Database
            |       Reload Database
            |       Back to Menu
```

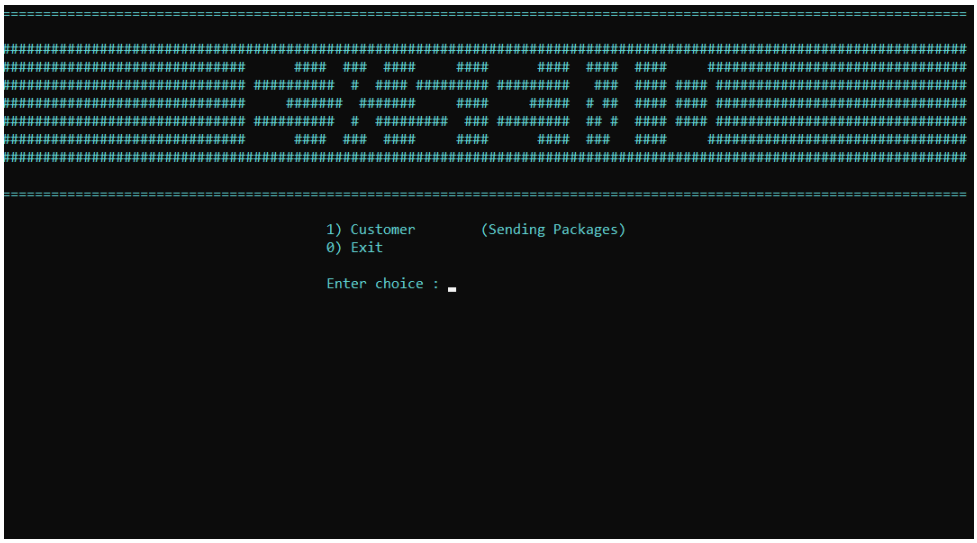
```
While Sender [ Status Check ]
        |       Show Package Status according to User Name
        Exit
        Back to Menu
    While Courier
        |       Show Menu
        |       Demand User Input
        |       Show Delivery Job
        Choose Delivery Job
            |       Run Dijkstra Algorithm source = delivery id
            |       Export Path to Memory
            |       Clear Linked List
            |       Export Read Path to Linked List
            |       Show Delivery Job Information and Route from Linked List
            |       Show Choices
            Change Status
                |       Status change to Ongoing - Cancelled – Delivered
                |       Location Input
                |       Export Data to Delivery Database
                |       Reload Database
            Next or Previous Route
                | Show Route according to choices
            Exit
    Show Main Menu
Exit
Export All Memory Data to Database
Program Ended
```

Penjelasan untuk *pseudocode* di atas sama dengan penjelasan untuk diagram alir pada bab sebelumnya.

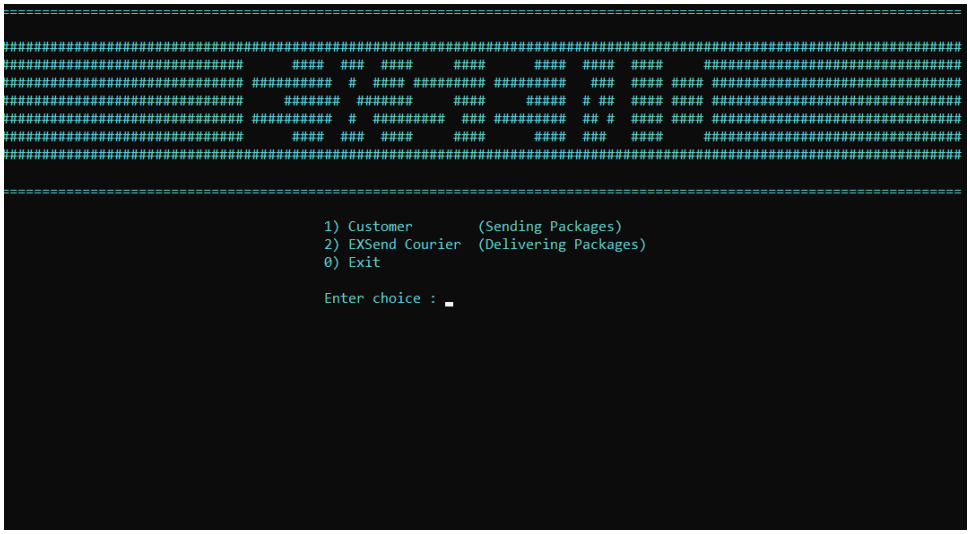
3.3 Tangkapan Layar Program



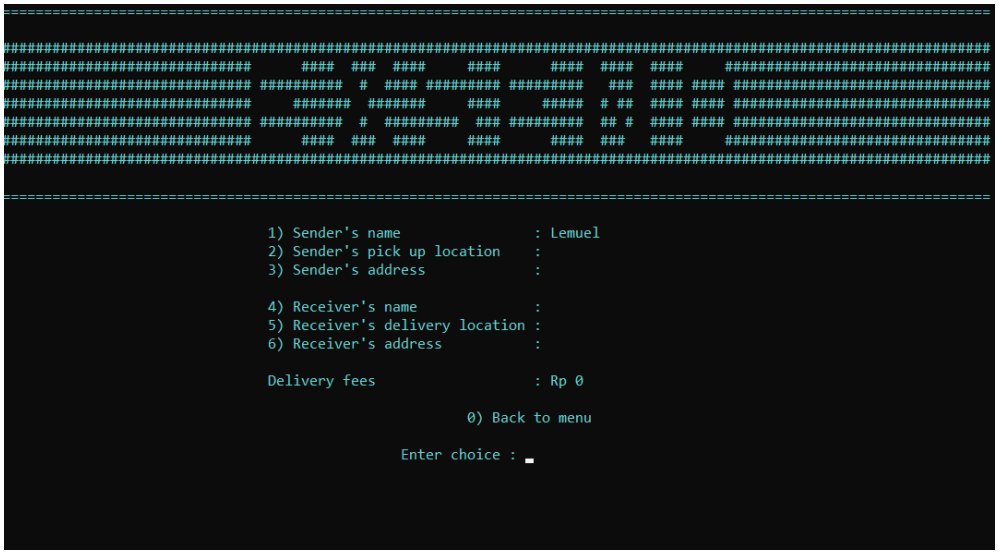
Layar utama jika pengguna berhasil *sign-in*.

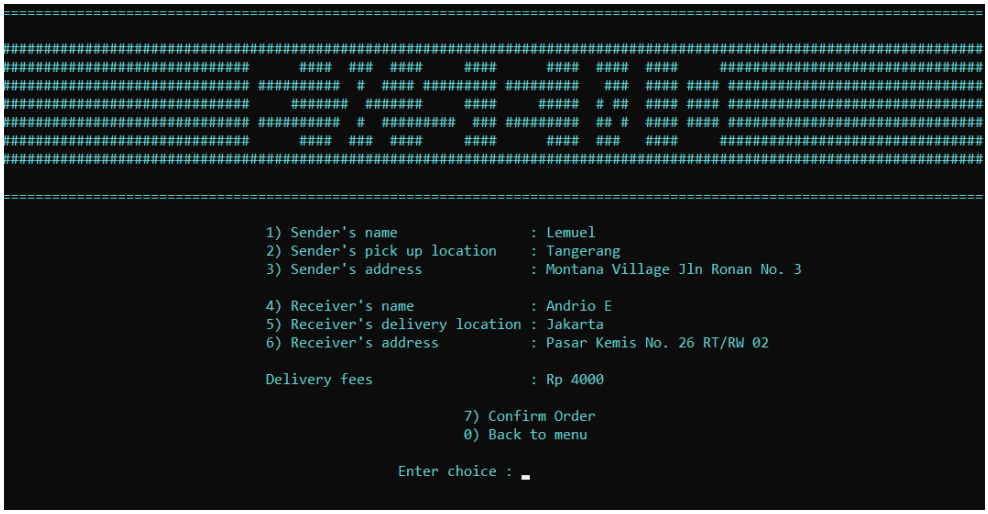


Tampilan layar apabila pengguna berhasil *sign-in* dan memiliki status sebagai *customer*.

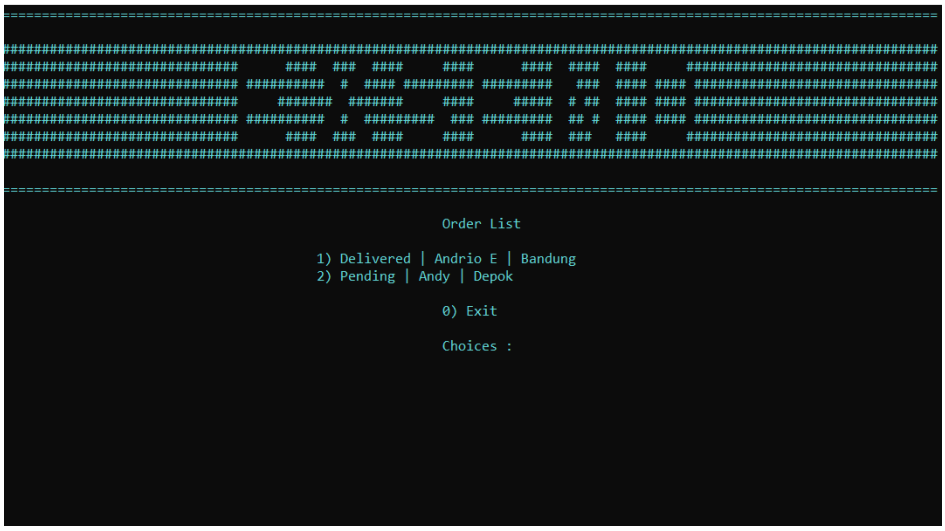


Tampilan layar apabila pengguna berhasil *sign-in* dan memiliki status sebagai *courier.*

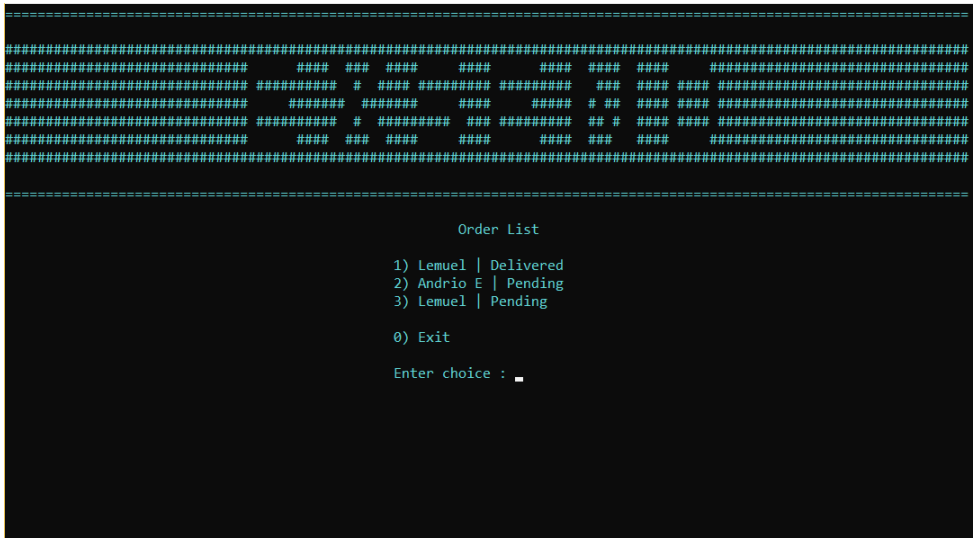




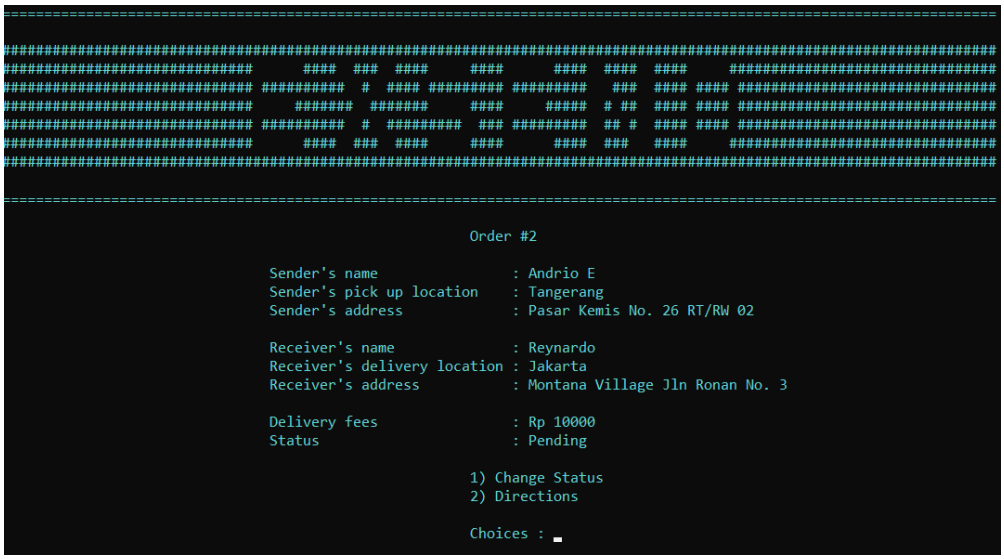
Kedua gambar di atas menampilkan layar pada saat pengguna memilih untuk mengirimkan paket, program akan meminta berbagai *input* (*sender's name, sender's pick up location, sender's address, receiver's name, receiver's delivery location, receiver's address)* dan seperti dapat dilihat, *delivery fees* otomatis diperbarui saat data yang dimiliki sudah lengkap. Saat kota dari pengirim dan penerima telah dimasukkan, algoritma Djikstra akan segera dijalankan untuk selanjutnya melakukan kalkulasi ongkos kirim.



Gambar di atas merupakan tampilan layar apabila pengguna yang berstatus sebagai *customer* melakukan pengecekan status pengiriman.



Sedangkan gambar di atas ini adalah daftar pengiriman yang statusnya dapat diubah oleh *courier*. *Courier* memiliki akses untuk mengubah status dari pesanan yang sudah tercatat dalam database. Untuk pilihan mengubah status akan dilampirkan pada gambar berikutnya.



Pilihan pertama berfungsi untuk mengubah status pesanan, sedangkan pilihan kedua berfungsi untuk menampilkan rute pengiriman. Dalam menu *Directions* ada dua pilihan lagi yang tersedia, yaitu menu rute selanjutnya dan sebelumnya. Menu ini bertujuan untuk memberikan arahan bagi kurir dalam melakukan pengiriman barang.

BAB IV
KESIMPULAN

Di era digital sekarang ini, kebutuhan masyarakat tentunya semangkit meningkat diiringi dengan makin padatnya kegiatan, waktu menjadi hal yang sangat berharga dan penting bagi masyarakat modern sehingga lingkungan masyarakat menuntut segala sesuatu untuk menjadi serba cepat. Hal ini tentu menjadi masalah di berbagai bidang pekerjaan termasuk jasa pengiriman barang. Jasa pengiriman barang konvensional biasanya tidak memiliki efisiensi waktu, seperti pengirim harus datang ke tempat jasa pengiriman itu, lalu memprosesnya. Tentu hal ini menguras tenaga dan waktu, belum lagi barang yang dikirim butuh waktu beberapa hari untuk sampai ke penerima. Ditambah lagi dengan harga yang biasanya tinggi dan terkadang ada minimum jumlah barang yang harus dikirim.

Dengan aplikasi EXSend yang kami buat ini, masalah seperti yang telah disebutkan di atas dapat diatasi. Di aplikasi ini, pengirim dapat melakukan pengisian data kapan saja dan di mana saja. Selain itu, kurir juga yang akan datang ke tempat pengambilan barang yang telah dimasukkan oleh pengirim untuk mengambil barang tersebut membuat pengirim tidak harus pergi ke kantor pos atau semacamnya. Hal ini dapat memangkas waktu sehingga pengirim dapat menggunakan waktu mereka dengan lebih efisien. Barang yang sudah diterima kurir langsung dikirimkan, yang memungkinkan untuk sampai dalam satu hari tanpa keterlambatan serta tidak perlu menambah biaya lagi. Tentunya hal ini sangat meringankan bagi pengusaha-pengusaha kecil dan menengah serta *start-up* baru yang ingin mengembangkan bisnis mereka.

DAFTAR PUSTAKA

Printing Paths in Dijkstra's Shortest Path Algorithm. (2019, February 27). Retrieved from https://www.geeksforgeeks.org/printing-paths-dijkstras-shortest-path-algorithm/

Tips dan trik menulis proposal skripsi bagi mahasiswa IT yang masih awam (Part I). (2012, February 14). Retrieved from https://janeman.wordpress.com/2011/11/10/tips-dan-trik-menulis-proposal-skripsi-bagi-mahasiswa-it-yang-masih-awam-part-i/

DAFTAR LAMPIRAN

Source Code EXSend

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////                    EXSend Software                    //////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/// For more information check EXSend Software Documentation

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <malloc.h>
#include <string.h>
#include <limits.h>
#include <stdbool.h>
#include <windows.h>

///---------------------------------------Global Variabel---------------------------------------///

// Struct Declare
struct route {
    char location_route[1][999];
    struct route *next, *prev;
};

// Variable Declare
int choices;
int global_id;

double graph_check = 0;
double graph_count = 0;

// User Input
int input_check = 0;

int user_privelege = 0;

char input_user[1][999];
char input_password[1][999];
char input_sender_name[1][999];
char input_sender_address[1][999];
char input_sender_location[1][999];
char input_receiver_name[1][999];
char input_receiver_address[1][999];
char input_receiver_location[1][999];
int input_price = 0;

// User Info
int login_check = 0;
int user_count = 0;
int user_id[999];
int privilege[999];

char user[999][999];
char password[999][999];

// Order Info
int order_count = 0;
int user_order_count = 0;
int user_order_id[1][999];
int order_id[999];
char order_status[999][999];
int node_count = 0;
int source = 0;
int destination = 0;
int graph[999][999];
int price_info[999];
char sender_name[999][999];
char receiver_name[999][999];
char sender_address[999][999];
```

```
67  char pickup_location[999][999];
68  char receiver_address[999][999];
69  char receiver_location[999][999];
70
71  // Location Info
72  int location_count = 0;
73  int sord = 0;
74  int location_id[999];
75  char location[999][999];
76
77  // Dijkstra Info
78  int d_count = 0;
79  int p_count = 0;
80  int length[999];
81  int distance[999];
82  int route[999][999];
83  char route_conv[999][999];
84
85  ///-------------------------------------------------------------------------------Startup-----------------------------
86
87  /// Load Database to Memory
88  void load_database()
89  {
90      // Clear Screen
91      clrscr();
92
93      // Database File
94      FILE *user_db;       // User Database
95      FILE *order_db;      // order Database
96      FILE *location_db;   // location Name Database
97      FILE *graph_db;      // Distance Graph Database
```

```
99      //Declare Variable
100     int counter_x = 0;
101     int counter_y = 0;
102
103     // Set file loader
104     user_db     = fopen("user.db", "r");
105     order_db    = fopen("order.db", "r");
106     location_db = fopen("location.db", "r");
107     graph_db    = fopen("graph.db", "r");
108
109 ///-------------------------------------------------------------------Check for file availability----------------
110
111     // Check for user.db availability
112     if (user_db == NULL)
113     {
114         printf("\n");
115         printf("Error while opening user.db!");
116         printf("\n");
117         exit(5);
118     }
119
120     // Check for order.db availability
121     if (order_db == NULL)
122     {
123         printf("\n");
124         printf("Error while opening order.db!");
125         printf("\n");
126         exit(5);
127     }
```

```
129     // Check for location.db availability
130     if (location_db == NULL)
131     {
132         printf("\n");
133         printf("Error while opening location.db!");
134         printf("\n");
135         exit(5);
136     }
137
138     // Check for graph.db availability
139     if (graph_db == NULL)
140     {
141         printf("\n");
142         printf("Error while opening graph.db!");
143         printf("\n");
144         exit(5);
145     }
146
147 ///-------------------------------------------------------------------Read File to Memory----------------------
148
149     // Read User Database to Memory
150     while ((fscanf(user_db, "%d#%[^\#]#%[^\#]#%d", &user_id[user_count], &user[user_count], &password[user_count],
151     {
152         user_count++;
153     }
```

```
    // Read order Database to Memory
    while ((fscanf(order_db, "%d#%[^\#]#%[^\#]#%[^\#]#%[^\#]#%[^\#]#%[^\#]#%[^\#]#%d",
        &order_id[order_count], &order_status[order_count], &sender_name[order_count], &receiver_name[order_count],
    {
        order_count++;
    }

    // Read Location Database to Memory
    while ((fscanf(location_db, "%d-%[^\n]", &location_id[location_count], &location[location_count])) != EOF)
    {
        location_count++;
        node_count++;
    }

    // Read Distance Graph Database to Memory
    while ((fscanf(graph_db, "%d ", &graph[counter_y][counter_x])) != EOF)
    {
        counter_x++;
        graph_count++;
        if (counter_x > location_count - 1)
        {
            counter_y++;
            counter_x = 0;
        }
    }

    // Checker to make sure the number of location in a file corresponding to the number of nodes in graph
    graph_check += sqrt(graph_count);
```

```
    if(graph_check != location_count)
    {
        printf("Invalid graph or location database!");
        printf("\n");
        exit(6);
    }
    if(counter_y != location_count)
    {
        printf("Invalid graph or location database!");
        printf("\n");
        exit(6);
    }

    // Close file
    fclose(user_db);
    fclose(order_db);
    fclose(location_db);
    fclose(graph_db);

}
```

```
void splash()
{
    // Print EXSend Logo
    printf("#################################################################################################################################################\n");
    printf("#################################################################################################################################################\n");
    printf("#################################################################################################################################################\n");
    printf("################################## ##############################      ###########################################################################\n");
    printf("################################  ############################      ###########################################################################\n");
    printf("##############################    ##########################      ### #########################################################################\n");
    printf("################################      ########################      ###  ########################################################################\n");
    printf("##################################        ######        ########      ###   ########################################################################\n");
    printf("##################################        ######        #####      ###    ########################################################################\n");
    printf("#####################################       ######       ###      ###    ########################################################################\n");
    printf("#####################################       ######      #      ###    ########################################################################\n");
    printf("#####################################         ######          ###     ########################################################################\n");
    printf("##############################################         #####    ###      ########################################################################\n");
    printf("##############################################       ##############      ########################################################################\n");
    printf("##############################################       #####   ##########       ########################################################################\n");
    printf("#############################################       #####        #########       ########################################################################\n");
    printf("#####################################       #####     #      #########       ########################################################################\n");
    printf("#####################################       #####      ###       #########       ########################################################################\n");
    printf("#####################################     #####       #####        #########       ########################################################################\n");
    printf("#####################################   #####       ########          #########       ########################################################################\n");
    printf("#####################################  #####       ###########################################################################################\n");
    printf("#########################################        ###########################################################################################\n");
    printf("#########################################       ###########################################################################################\n");
    printf("#########################################       ###########################################################################################\n");
    printf("#################################################################################################################################################\n");
    printf("#################################################################################################################################################\n");
    printf("#################################################################################################################################################\n");
}
```

```
void banner()
{
    // Print banner
    printf("=============================================================================================================================");
    printf("\n");
    printf("#############################################################################################################################");
    printf("################################      ####  ###  ####     ####      ####  ####  ####     ########################");
    printf("################################ ##########  #  #### ######### #########   ###  #### #### ########################");
    printf("################################     #######  #######     ####     #####  # ##  #### #### ########################");
    printf("################################ ##########  #  #########  ### #########  ## #  #### #### ########################");
    printf("################################      ####  ###  ####     ####      ####  ###   ####     ########################");
    printf("#############################################################################################################################");
    printf("\n");
    printf("=============================================================================================================================");
    printf("\n");
}

void login()
{
    // Variable Set
    global_id = 1;
    login_check = 0;

    // Clear Screen
    clrscr();

    // Show banner
    banner();

    // Print Login Menu
    printf("\t\t\t\t\t\t");   printf("Username : ");                                scanf("%[^\n]",
    printf("\t\t\t\t\t\t");   printf("Password : ");                                scanf("%[^\n]",

    /// Check Username and Password
    search(input_user);

    if(login_check == 1)
    {
        global_id = 2;
        login_check = 0;
        search(input_password);
    }

    // If Username or Password is incorrect
    if(login_check == 0)
    {
        printf("\t\t\t\t\t\t"); printf("Incorrect Username or Password!");          printf("\n");
        printf("\t\t\t\t\t\t"); printf("Press any key to continue...");             printf("\n");
        getch();
        login();
    }

    // If Username and Password is correct
    if(login_check == 1){
        printf("\t\t\t\t\t\t"); printf("Welcome back %s!", input_user);             printf("\n");
        printf("\t\t\t\t\t\t"); printf("Press any key to continue...", input_user); getch();
        main_menu();
    }

}

void main_menu()
{
    // Variable Set
    global_id = 3;

    // Clear Screen
    clrscr();

    // Show banner
    banner();

    // Print Menu
    printf("\t\t\t\t\t\t"); printf("1) Customer        (Sending Packages)");       printf("\n");

    //If Courier show courier menu
    if(user_privelege == 2)
    {
        printf("\t\t\t\t\t\t"); printf("2) EXSend Courier  (Delivering Packages)"); printf("\n");
    }

    printf("\t\t\t\t\t\t"); printf("0) Exit");                                     printf("\n");
    printf("\n");
    printf("\t\t\t\t\t\t"); printf("Enter choice : ");                            scanf("%d", &choices);

    // Check for user choices
    switch(choices)
    {
        case 1 :
        {
            menu_customer();
            break;
        }
```

```
332             case 2 :
333             {
334                 // Make sure normal users can access courier menu
335                 if(user_privelege == 2)
336                 {
337                     menu_courier();
338                 }
339                 else
340                 {
341                     invalid();
342                     main_menu();
343                 }
344                 break;
345             }
346
347             case 0 :
348             {
349                 exit(0);
350             }
351         }
352     }
```

```
354     void menu_customer()
355     {
356         // Variable Set
357         global_id = 4;
358
359         // Clear Screen
360         clrscr();
361
362         // Show banner
363         banner();
364
365         // Show Menu
366         printf("\t\t\t\t\t\t"); printf("1) EXSend Delivery");            printf("\n");
367         printf("\t\t\t\t\t\t"); printf("2) Delivery status check");     printf("\n");
368         printf("\t\t\t\t\t\t"); printf("0) Back");                       printf("\n");
369         printf("\n");
370         printf("\t\t\t\t\t\t"); printf("Enter choice : ");              scanf("%d", &choices);  getchar();
371
372         // Check for user choices
373         switch(choices)
374         {
375             case 1 :
376             {
377                 customer_delivery();
378                 break;
379             }
380
381             case 2 :
382             {
383                 menu_ustatus();
384                 break;
385             }
```

```
387             case 0 :
388             {
389                 main_menu();
390                 break;
391             }
392         }
393     }
```

```
void customer_delivery()
{
    // Set Global ID
    global_id = 41;

    // Clear Screen
    clrscr();

    // Show banner
    banner();

    /// Input Check
    if(input_check >= 5)
    {
        dijkstra();
        input_price = ( distance[destination] * 1000 );
    }

    printf("\t\t\t\t\t"); printf("1) Sender's name                : %s", user);
    printf("\t\t\t\t\t"); printf("2) Sender's pick up location    : %s", input_sender_location);
    printf("\t\t\t\t\t"); printf("3) Sender's address             : %s", input_sender_address);
    printf("\n");
    printf("\t\t\t\t\t"); printf("4) Receiver's name              : %s", input_receiver_name);
    printf("\t\t\t\t\t"); printf("5) Receiver's delivery location : %s", input_receiver_location);
    printf("\t\t\t\t\t"); printf("6) Receiver's address           : %s", input_receiver_address);

    printf("\n");

    printf("\t\t\t\t\t"); printf("Delivery fees                  : Rp %d", input_price);

    // Print Confirm Order if users filled in the data
    if(input_check >= 5)
    {
        printf("\n");
        printf("\t\t\t\t\t\t\t\t");  printf("7) Confirm Order");
    }

    printf("\n");
    printf("\t\t\t\t\t\t\t\t");      printf("0) Back to menu");

    printf("\n");

    printf("\t\t\t\t\t\t\t\t");      printf("Enter choice : ");

    // If user decided to confirm and to make sure users are not able to choose 7 before filled in th
    if(input_check >= 5)
    {
        if(choices == 7)
        {
            // Export to file
            export_tofile();

            // Reset variable
            reset();

            // Re-Load database
            load_database();

            // Show users that the order has been placed
            printf("\t\t\t\t\t\t\t"); printf("Your order has been recorded!");
            printf("\t\t\t\t\t\t\t"); printf("We will called you when we are ready."); printf("\n");

            // Back to menu
            menu_customer();
        }
    }

    switch(choices)
    {
        case 1 :
        {
            delivery_name();
            break;
        }

        case 2 :
        {
            delivery_location();
            break;
        }

        case 3 :
        {
            delivery_address();
            break;
        }

        case 4 :
        {
            delivery_name();
            break;
        }
```

```
488             case 5 :
489             {
490                 delivery_location();
491                 break;
492             }
493
494             case 6 :
495             {
496                 delivery_address();
497                 break;
498             }
499
500             case 0 :
501             {
502                 menu_customer();
503             }
504         }
505
506     }
```

```
508     void delivery_name()
509     {
510         global_id = 411;
511
512         switch(choices)
513         {
514             case 4 :
515             {
516                 // Clear Screen
517                 clrscr();
518
519                 // Show banner
520                 banner();
521
522                 // Demand user input
523                 printf("\t\t\t\t\t\t"); printf("Receiver's name               : "); scanf("%[^\n]",
524
525                 // Back to delivery menu
526                 input_check++;
527                 customer_delivery();
528                 break;
529             }
530         }
531
532     }
```

```
534     void delivery_location()
535     {
536         global_id = 412;
537
538         int count;
539
540         switch(choices)
541         {
542             case 2 :
543             {
544                 // Clear Screen
545                 clrscr();
546
547                 // Show banner
548                 banner();
549
550                 // Print all available location
551                 for(count = 0; count < location_count; count++)
552                 {
553                     printf("\t\t\t\t\t\t"); printf("%d) %s", count+1, location[count]);        printf("\n
554                 }
555
556                 // Demand user input
557                 printf("\n");
558                 printf("\t\t\t\t\t\t");     printf("Choose pickup location : ");              scanf("%d"
559
560                 // Choices to IDs converter
561                 choices--;
562                 strcpy(input_sender_location, location[choices]);
563
564                 // Set source to user_input
565                 source = choices;
```

```
            // Back to delivery menu
            input_check++;
            customer_delivery();
            break;
        }

        case 5 :
        {
            // Clear Screen
            clrscr();

            // Show banner
            banner();

            // Print all available location
            for(count = 0; count < location_count; count++)
            {
                printf("\t\t\t\t\t"); printf("%d) %s", count+1, location[count]);     printf("\n");
            }

            printf("\n");

            // Demand user input
            printf("\t\t\t\t\t");     printf("Choose receiver's location : ");        scanf("%d", &choices);

            // Choices to IDs converter
            choices--;
            strcpy(input_receiver_location, location[choices]);

            // Set destination to user_input
            destination = choices;

             // Back to delivery menu
             input_check++;
             customer_delivery();
             break;
         }
    }
}

void delivery_address()
{
    global_id = 413;

    switch(choices)
    {
        case 3 :
        {
            // Clear Screen
            clrscr();

            // Show banner
            banner();

            printf("\t\t\t\t"); printf("Sender's Address                 : ");     scanf("%[^\n]", input_sender_

            // Back to delivery menu
            input_check++;
            customer_delivery();
            break;
        }

        case 6 :
        {
            // Clear Screen
            clrscr();

            // Show banner
            banner();

            printf("\t\t\t\t"); printf("Receiver's Address               : ");     scanf("%[^\n]", input_receive

             // Back to delivery menu
             input_check++;
             customer_delivery();
             break;
         }
    }
}
```

```c
void menu_ustatus()
{
    // Variable Set
    global_id = 42;

    // Clear Screen
    clrscr();

    // Print Banner
    banner();

    // Print Menu
    printf("\t\t\t\t\t\t\t\t");     printf("Order List");    printf("\n");

    printf("\n");

    // Show only users order
    search(user);

    printf("\n");

    // Print Exit option
    printf("\t\t\t\t\t\t\t\t");     printf("0) Exit");       printf("\n");

    // Reset order_count
    user_order_count = 0;

    printf("\n");

    // Demand user input
    printf("\t\t\t\t\t\t\t\t");     printf("Choices : ");    scanf("%d", &choices);

    order_show();
}

void order_show()
{
    // Declare
    int id = choices-1;

    // Clear Screen
    clrscr();

    // Print Banner
    banner();

    // Convert choices to order IDs
    id = user_order_id[1][id];

    // Back to menu
    if(choices == 0)
    {
        menu_customer();
    }

    // Print Order Information
    printf("\t\t\t\t\t\t\t\t"); printf("Order #%d", choices);

    printf("\n");

    printf("\t\t\t\t\t");       printf("Sender's name                : %s", sender_name[id]);
    printf("\t\t\t\t\t");       printf("Sender's pick up location    : %s", pickup_location[id]);
    printf("\t\t\t\t\t");       printf("Sender's address             : %s", sender_address[id]);
    printf("\n");
    printf("\t\t\t\t\t");       printf("Receiver's name              : %s", receiver_name[id]);
    printf("\t\t\t\t\t");       printf("Receiver's delivery location : %s", receiver_location[id]);
    printf("\t\t\t\t\t");       printf("Receiver's address           : %s", receiver_address[id]);

    printf("\n");

    printf("\t\t\t\t\t");       printf("Delivery fees                : Rp %d", price_info[id]);
    printf("\t\t\t\t\t");       printf("Status                       : %s", order_status[id]);

    printf("\t\t\t\t\t");       printf("Press any key to go back...");

    // Back to user order status
    menu_ustatus();

}
```

```
729  void menu_courier()
730  {
731      // Variable Set and Declare
732      int counter;
733      global_id = 5;
734
735      // Clear Screen
736      clrscr();
737
738      // Show banner
739      banner();
740
741      // Print Menu
742      printf("\t\t\t\t\t\t\t\t");    printf("Order List");
743      printf("\n");
744
745      for(counter = 0; counter < order_count; counter++)
746      {
747          printf("\t\t\t\t\t\t\t");  printf("%d) %s | %s", counter+1, sender_name[counter], order_status[counter]);
748      }
749
750      printf("\n");
751
752      // Print exit option
753      printf("\t\t\t\t\t\t\t");      printf("0) Exit");
754
755      printf("\n");
756
757      // Demand user input
758      printf("\t\t\t\t\t\t\t");      printf("Enter choice : ");

760      // Exit to main menu
761      if(choices == 0)
762      {
763          main_menu();
764      }
765
766      // Check for user choices
767      delivery_show();
768  }

770  void delivery_show()
771  {
772      // Declare variable and set
773      int input;
774      global_id = 51;
775      input = choices;
776      input--;
777
778      // Clear Screen
779      clrscr();
780
781      // Show banner
782      banner();
783
784      // Print Order Information
785      printf("\t\t\t\t\t\t\t");   printf("Order #%d", input+1);
786
787      printf("\n");
788
789      printf("\t\t\t\t");         printf("Sender's name                : %s", sender_name[input]);
790      printf("\t\t\t\t");         printf("Sender's pick up location    : %s", pickup_location[input]);
791      printf("\t\t\t\t");         printf("Sender's address             : %s", sender_address[input]);
792      printf("\n");
793      printf("\t\t\t\t");         printf("Receiver's name              : %s", receiver_name[input]);
794      printf("\t\t\t\t");         printf("Receiver's delivery location : %s", receiver_location[input]);
795      printf("\t\t\t\t");         printf("Receiver's address           : %s", receiver_address[input]);
796
797      printf("\n");
798
799      printf("\t\t\t\t");         printf("Delivery fees                : Rp %d", price_info[input]);
800      printf("\t\t\t\t");         printf("Status                       : %s", order_status[input]);
801
802      printf("\n");
```

```
    printf("\t\t\t\t\t\t\t\t");   printf("1) Change Status");
    printf("\t\t\t\t\t\t\t\t");   printf("2) Directions");
    printf("\t\t\t\t\t\t\t\t");   printf("0) Back to menu");

    printf("\n");

    printf("\t\t\t\t\t\t\t\t");   printf("Choices : ");

    switch(choices)
    {
        case 1 :
        {
            menu_cstatus(input);
        }
        case 2 :
        {
            show_direction(input);
        }
        case 0 :
        {
            menu_courier();
        }
    }
}

void menu_cstatus(int *input)
{
    // Declare variable and set
    global_id = 511;
    int id;

    id = input;
    id++;

    // Clear Screen
    clrscr();

    // Show banner
    banner();

    printf("\t\t\t\t\t\t\t\t");           printf("Order #%d", id);
    printf("\t\t\t\t\t\t      ");         printf("Change Status");

    printf("\n");

    printf("\t\t\t\t\t\t      ");         printf("1) Pending");
    printf("\t\t\t\t\t\t      ");         printf("2) Delivered");

    for(int count = 0; count < node_count; count++)
    {
        printf("\t\t\t\t\t\t      ");     printf("%d) %s", count+3, location[count]);
    }

    printf("\n");

    printf("\t\t\t\t\t\t\t\t");           printf("Choices : ");

    id--;

    // Modify corresponding data in memory
    modify_memory(id);

    // Back to courier menu
    menu_courier();
}

void search(char *search)
{
    // Declare variable
    int mid, low, high;
    low = 0;
    int counter = 0;

    // Set high variable according to global id
    if(global_id == 1)
    {
        high = user_count - 1;
    }

    if(global_id == 2)
    {
        high = user_count - 1;
    }

    if(global_id == 51)
    {
        high = location_count - 1;
    }
```

```
894     /// Sequential Search
895     switch(global_id)
896     {
897         case 1 :
898         {
899             for(counter = 0; counter < user_count; counter++)
900             {
901                 if (strcmpi(search, user[counter]) == 0)
902                     {
903                         login_check = 1;
904                     }
905             }
906             break;
907         }
908
909         case 2 :
910         {
911             for(counter = 0; counter < user_count; counter++)
912             {
913                 if (strcmpi(search, password[counter]) == 0)
914                     {
915                         login_check = 1;
916                         user_privelege = privilege[counter];
917                     }
918             }
919             break;
920         }
```

```
922         case 51 :
923         {
924             for(counter = 0; counter < location_count; counter++)
925             {
926                 if (strcmpi(search, location[counter]) == 0)
927                     {
928                         if(sord == 0)
929                         {
930                             source = counter;
931                         }
932                         if(sord == 1)
933                         {
934                             destination = counter;
935                         }
936                     }
937             }
938             break;
939         }
940
941         case 42 :
942         {
943             for(counter = 0; counter < order_count; counter++)
944             {
945                 if (strcmpi(search, sender_name[counter]) == 0)
946                     {
947                         user_order_id[1][user_order_count] = counter;
948                         user_order_count++;
949                         printf("\t\t\t\t\t"); printf("%d) %s | %s | %s", user_order_count, order_stat
950                     }
951             }
952             break;
953         }
954     }
955 }
```

```
957 /// Dijkstra Algorithm
958 int minDistance(int dist[],bool sptSet[])
959 {
960
961     // Initialize min value
962     int min = INT_MAX, min_index;
963
964     for (int v = 0; v < node_count; v++)
965         if (sptSet[v] == false && dist[v] <= min)
966             min = dist[v], min_index = v;
967
968     return min_index;
969 }
```

```
971  void export_route(int parent[], int j)
972  {
973      if (parent[j] == - 1)
974      {
975          return;
976      }
977
978      // Recursive
979      export_route(parent, parent[j]);
980
981      // Store path to memory
982      route[d_count][p_count] = j;
983
984      // Counter
985      p_count++;
986
987  }
```

```
989  int export_dijkstra(int dist[], int n, int parent[])
990  {
991      for (int i = 0; i < node_count; i++)
992      {
993          // Store distance to memory
994          distance[i] = dist[i];
995
996          // Add Source to path
997          route[d_count][p_count] = source;
998          p_count++;
999
1000         // Store how much path are in memory
1001         length[i] = p_count;
1002
1003         // Export path to memory
1004         export_route(parent, i);
1005
1006         // Store how much path are in memory
1007         length[i] = p_count;
1008
1009         // Reset and counter
1010         p_count = 0;
1011         d_count++;
1012     }
1013 }
```

```
1015 void dijkstra()
1016 {
1017     // store shortest distance
1018     int dist[node_count];
1019
1020     // Declare sptSet
1021     bool sptSet[node_count];
1022
1023     // Store shortest path tree
1024     int parent[node_count];
1025
1026     // Initialize all distances as infinite and set sptSet to false
1027     for (int i = 0; i < node_count; i++)
1028     {
1029         parent[source] = -1;
1030         dist[i] = INT_MAX;
1031         sptSet[i] = false;
1032     }
1033
1034     // Set distance from source to source is 0
1035     dist[source] = 0;
1036
1037     // Find shortest path from all vertex
1038     for (int count = 0; count < node_count - 1; count++)
1039     {
1040         // Choose minimum distance
1041         int u = minDistance(dist, sptSet);
1042
1043         // Set the chosen vertex to true
1044         sptSet[u] = true;
```

```
1046            // Update distance value
1047            for (int v = 0; v < node_count; v++)
1048
1049                // Update if not in sptSet
1050                if (!sptSet[v] && graph[u][v] &&
1051                    dist[u] + graph[u][v] < dist[v])
1052                {
1053                    parent[v] = u;
1054                    dist[v] = dist[u] + graph[u][v];
1055                }
1056        }
1057
1058        // Export result to memory
1059        export_dijkstra(dist, node_count, parent);
1060    }
```

```
1062    void modify_memory(int *id)
1063    {
1064        // Declare and set variable
1065        int local_id;
1066        local_id = id;
1067
1068        switch(global_id)
1069        {
1070            case 511 :
1071            {
1072                if(choices == 1)
1073                {
1074                    strcpy(order_status[local_id], "Pending");
1075                    break;
1076                }
1077                if(choices == 2)
1078                {
1079                    strcpy(order_status[local_id], "Delivered");
1080                    break;
1081                }
1082                else
1083                {
1084                    choices = choices - 3;
1085                    strcpy(order_status[local_id], location[choices]);
1086                    break;
1087                }
1088                break;
1089            }
1090        }
```

```
1093    void export_tofile()
1094    {
1095        // Database File
1096        FILE *user_db;      // User Database
1097        FILE *order_db;     // order Database
1098        FILE *location_db;  // location Name Database
1099        FILE *graph_db;     // Distance Graph Database
1100
1101        // Set file loader and open file
1102        user_db = fopen("user.db", "a");
1103        order_db = fopen("order.db", "a");
1104        location_db = fopen("location.db", "a");
1105        graph_db = fopen("graph.db", "a");
1106
1107        switch(global_id)
1108        {
1109            case 41 :
1110            {
1111                fprintf(order_db, "\n%d#Pending#%s#%s#%s#%s#%s#%s#%d",
1112                        order_count, user, input_receiver_name, input_sender_address, input_sender_location,
1113                break;
1114            }
1115        }
1116
1117        // Close file
1118        fclose(user_db);
1119        fclose(order_db);
1120        fclose(location_db);
1121        fclose(graph_db);
1122    }
```

```
1124  void overwrite_tofile()
1125  {
1126      // Declare variable
1127      int counter;
1128
1129      // Database File
1130      FILE *user_db;       // User Database
1131      FILE *order_db;      // order Database
1132      FILE *location_db;   // location Name Database
1133      FILE *graph_db;      // Distance Graph Database
1134
1135      // Set file loader
1136      if(global_id == 511)
1137      {
1138          order_db = fopen("order.db", "w");
1139      }
1140
1141      // Overwrite database from memory
1142      switch(global_id)
1143      {
1144          case 511 :
1145          {
1146              for(counter = 0; counter < order_count; counter++)
1147              {
1148                  fprintf(order_db, "%d#%s#%s#%s#%s#%s#%s#%d\n",
1149                      order_id[counter], order_status[counter], sender_name[counter], receiver_name[counter]
1150              }
1151              break;
1152          }
1153      }
1154
1155      // Close file
1156      fclose(order_db);
1157  }
```

```
1159  void show_direction(int *input)
1160  {
1161      // Declare Variable
1162      int local_input;
1163      int counter = 0;
1164      int insert = 0;
1165      int page = 1;
1166      sord = 0;        // 0 = Set to source || 1 = Set to destination (Used to set
1167
1168      local_input = input;
1169      struct route *head, *node, *tail, *curr;
1170
1171      head = NULL;
1172
1173      // Read Array to determine source and destination
1174      search(pickup_location[local_input]);
1175      sord++;
1176      search(receiver_location[local_input]);
1177
1178      // Run Dijkstra Algorithm
1179      dijkstra();
1180
1181      // Add Linked List
1182      for(counter = 0; counter < length[destination]; counter++)
1183      {
1184          node = (struct route*) malloc(sizeof(struct route));
1185          node->next = NULL;
1186          node->prev = NULL;
1187
1188          insert =  route[destination][counter];
1189          strcpy(node->location_route, location[insert]);
```

```
1191        if(head == NULL)
1192        {
1193            head = node;
1194            tail = node;
1195        }
1196        else{
1197            tail->next = node;
1198            node->prev = tail;
1199            tail = node;
1200        }
1201    }
1202
1203    curr = head;
1204
1205    // Print Directions
1206    while(1)
1207    {
1208        // Clear Screen
1209        clrscr();
1210
1211        // Show banner
1212        banner();
1213
1214        printf("\t\t\t\t\t\t\t    ");          printf("=== Order #%d ===", local_input+1);
1215        printf("\n");
1216        printf("\t\t\t\t\t\t\t      ");        printf("Direction #1", local_input+1);
1217        printf("\n");
1218        printf("\t\t\t\t\t\t\t       ");       printf("%s", curr->location_route);
1219
1220        printf("\n");
```

```
1220        printf("\n");
1221
1222        printf("\t\t\t\t      ");            printf("1) Previous Direction");
1223        printf("\n");
1224        printf("\t\t\t\t\t\t\t\t\t");          printf("0) Exit");
1225        printf("\n");
1226        printf("\t\t\t\t\t\t\t       ");      printf("Choices : ");
1227
1228        if(choices == 2)
1229        {
1230            if(curr->next == NULL)
1231            {
1232                printf("\t\t\t\t\t\t    ");   printf("Already at the final destination");
1233                getch();
1234            }
1235            else
1236            {
1237                curr = curr->next;
1238            }
1239        }
1240        else if(choices == 1)
1241        {
1242            if(curr->prev == NULL)
1243            {
1244                printf("\t\t\t\t\t\t ");      printf("Already at the start of the destination");
1245                getch();
1246            }
1247            else
1248            {
1249                curr = curr->prev;
1250            }
```

```
1252        else if(choices == 0)
1253        {
1254            // Clear Linked List
1255            while(head != NULL)
1256            {
1257                curr = head;
1258                head = head->next;
1259                free(curr);
1260            }
1261
1262            // Exit to courier menu
1263            menu_courier();
1264        }
1265    }
1266 }
```

```c
int main()
{
    // Set color to default
    system("MODE 120, 27");
    set_color();

    // Load database
    load_database();

    // Clear Screen
    clrscr();

    // Show EXSend splash logo animation
    splash();

    Sleep(500);
    system("COLOR 09");
    Sleep(500);
    system("COLOR 03");
    Sleep(500);
    system("COLOR 0B");
    Sleep(1000);

    // Clear Screen
    clrscr();

    // Sign in
    login();

    // Print Menu
    main_menu();
}
```

```c
void reset()
{
    // Reset Variable
    input_check = 0;
    user_count = 0;
    order_count = 0;
    location_count = 0;
    node_count = 0;
    graph_count = 0;
    graph_check = 0;
}

void set_color()
{
    // Black Background and white text
    system("COLOR 0F");
}

void invalid()
{
    // Print invalid input to users
    printf("\t\t\t\t\t\t"); printf("Invalid Input!");                printf("\n");
    printf("\t\t\t\t\t\t"); printf("Press any key to continue..."); getch();
}

void clrscr()
{
    system("cls");
}
```