1. Penjelasan Cara Kerja Code Program Circular Double Linked List:

    1) Struct Node

```
typedef struct Node {
2)     int data;
3)     struct Node *next;
4)     struct Node *prev;
5) } Node;
6)
```

Struct Node didefinisikan dengan dengan tiga bagian, yaitu *data* (yang menyimpan nilai), *next* (yang menunjuk ke node berikutnya), dan *prev (*yang menunjuk ke node sebelumnya).

2) Void CreateNode

```
Node* createNode(int data) {
    Node *newNode = (Node*)malloc(sizeof(Node));
    newNode->data = data;
    newNode->next = newNode->prev = newNode;
    return newNode;
}
```

Fungsi ini digunakan untuk membuat npde baru. Node baru dibuat dengan menggunakan fungsi *malloc* untuk mengalokasikan memori. Nilai data diset dan pointer *next* dan *prev* diatur untuk menujuk ke node itu sendiri.

3) Void insertNode

```
void insertNode(Node **head, int data) {
    Node *newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
    } else {
        Node *last = (*head)->prev;
        newNode->next = *head;
        (*head)->prev = newNode;
        newNode->prev = last;
        last->next = newNode;
    }
}
```

Fungsi ini digunakan untuk memasukan node bar uke dalam list. Jika list kosong, node baru menjadi kepala list atau head. Jika tidak, node baru disisipkan di antara node terakhir dan node kepala(head).

4) Fungsi PrintList

```
void printList(Node *head, const char* label) {
    printf("%s\n", label);
    Node *temp = head;
    char addressStr[20];
    do {
        sprintf(addressStr, "%p", (void *)temp);
        printf("Address: %s, Data: %d\n", addressStr, temp->data);
        temp = temp->next;
    } while(temp != head);
}
```

Fungsi ini digunakan untuk mencetak semua elemen dalam list. Fungsi ini mencetak alamat dan data setiap node.

5) Fungsi removeDuplicates

```
void removeDuplicates(Node **head) {
    if (*head == NULL) return;
    Node *current, *index, *temp;
    for (current = *head; current->next != *head; current = current->next) {
        for (index = current->next; index != *head; index = index->next) {
            if (current->data == index->data) {
                temp = index;
                index->prev->next = index->next;
                index->next->prev = index->prev;
                if (index == *head) *head = index->next;
                index = index->prev;
                free(temp);
            }
        }
    }
}
```

Fungsi ini digunakan untuk menghapus node-node yang memiliki

Nilai data yang sama. Fungsi ini melakukan iterasi melalui setiap node dan membandingkan dan membandingkan nilai data dengan setiap node lainnya dalam list.

6) Fungsi sortList

```
void sortList(Node **head) {
    if(*head == NULL) {
        return;
    }
```

```
    Node *current = NULL, *index = NULL;
    int temp;
    for(current = *head; current->next != *head; current = current->next) {
        for(index = current->next; index != *head; index = index->next) {
            if(current->data > index->data) {
                temp = current->data;
                current->data = index->data;
                index->data = temp;
            }
        }
    }
}
```

Fungsi ini digunakan untuk mengatur node-node dalam lits berdasarkan nilai data mereka. Fungsi ini menggukan algoritma bubble sort.

7) Fungsi main

```
int main() {
    Node *head = NULL;
    int n, data;
    printf("Masukkan jumlah data (1-10): ");
    scanf("%d", &n);
    for(int i = 0; i < n; i++) {
        printf("Masukkan data (1-10): ");
        scanf("%d", &data);
        insertNode(&head, data);
    }
    printList(head, "List sebelum pengurutan:");
    sortList(&head);
    removeDuplicates(&head);
    printList(head, "List setelah pengurutan:");
    return 0;
}
```

Fungsi ini adalah titik masuk program. Fungsi ini meminta pengguna untuk memasukkan jumlah data dan nilai data, menambahkan setiap data ke dalam daftar menggunakan *'insertNode'*. Setelah itu, daftar diceta, diurutan dengan '*sortList*' dan menghapus duplikat menggunakan '*removeDuplicates*'. Akhirnya, daftar yang sudah diurutkan dan tampa duplikat dicetak.

2. Source code

```c
#include <stdio.h>
#include <stdlib.h>

typedef struct Node {
    int data;
    struct Node *next;
    struct Node *prev;
} Node;

Node* createNode(int data) {
    Node *newNode = (Node*)malloc(sizeof(Node));
    newNode->data = data;
    newNode->next = newNode->prev = newNode;
    return newNode;
}

void insertNode(Node **head, int data) {
    Node *newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
    } else {
        Node *last = (*head)->prev;
        newNode->next = *head;
        (*head)->prev = newNode;
        newNode->prev = last;
        last->next = newNode;
    }
}

void printList(Node *head, const char* label) {
    printf("%s\n", label);
    Node *temp = head;
    char addressStr[20];

    do {
        sprintf(addressStr, "%p", (void *)temp);
        printf("Address: %s, Data: %d\n", addressStr, temp->data);
        temp = temp->next;
    } while(temp != head);
}


void removeDuplicates(Node **head) {
    if (*head == NULL) return;
```

```
    Node *current, *index, *temp;
    for (current = *head; current->next != *head; current = current->next) {
        for (index = current->next; index != *head; index = index->next) {
            if (current->data == index->data) {
                temp = index;
                index->prev->next = index->next;
                index->next->prev = index->prev;
                if (index == *head) *head = index->next;
                index = index->prev;
                free(temp);
            }
        }
    }
}

void sortList(Node **head) {
    if(*head == NULL) {
        return;
    }
    Node *current = NULL, *index = NULL;
    int temp;

    for(current = *head; current->next != *head; current = current->next) {
        for(index = current->next; index != *head; index = index->next) {
            if(current->data > index->data) {
                temp = current->data;
                current->data = index->data;
                index->data = temp;
            }
        }
    }
}

int main() {
    Node *head = NULL;
    int n, data;
    printf("Masukkan jumlah data (1-10): ");
    scanf("%d", &n);
    for(int i = 0; i < n; i++) {
        printf("Masukkan data (1-10): ");
        scanf("%d", &data);
        insertNode(&head, data);
    }
    printList(head, "List sebelum pengurutan:");
    sortList(&head);
```

```
    removeDuplicates(&head);
    printList(head, "List setelah pengurutan:");

    return 0;
}
```

3. Ss Output:

- Output pertama:

```
Masukkan jumlah data (1-10): 6
Masukkan data (1-10): 5
Masukkan data (1-10): 5
Masukkan data (1-10): 3
Masukkan data (1-10): 8
Masukkan data (1-10): 1
Masukkan data (1-10): 6
List sebelum pengurutan:
Address: 0000000000664ED0, Data: 5
Address: 0000000000664F20, Data: 5
Address: 0000000000664F70, Data: 3
Address: 00000000006615D0, Data: 8
Address: 0000000000661620, Data: 1
Address: 0000000000661670, Data: 6
List setelah pengurutan:
Address: 0000000000664ED0, Data: 1
Address: 0000000000664F20, Data: 3
Address: 0000000000664F70, Data: 5
Address: 0000000000661620, Data: 6
Address: 0000000000661670, Data: 8
```

- Output kedua:

```
Masukkan jumlah data (1-10): 4
Masukkan data (1-10): 3
Masukkan data (1-10): 31
Masukkan data (1-10): 2
Masukkan data (1-10): 123
List sebelum pengurutan:
Address: 0000000000B34ED0, Data: 3
Address: 0000000000B34F20, Data: 31
Address: 0000000000B34F70, Data: 2
Address: 0000000000B315D0, Data: 123
List setelah pengurutan:
Address: 0000000000B34ED0, Data: 2
Address: 0000000000B34F20, Data: 3
Address: 0000000000B34F70, Data: 31
Address: 0000000000B315D0, Data: 123
PS D:\Kuliah\Smester 2\Praktikum ASD>
```