

**Nama : Farrel Ahnaf Khayla Praptama**

**NIM : 1203230100**

**KLS : IF 03-03**

## **TUGAS CIRCULAR DOUBLE LINKED LIST**

### **SOURCE CODE**

```
#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 insertEnd(Node** head, int data) {
    Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
    } else {
        Node* tail = (*head)->prev;
        tail->next = newNode;
        newNode->prev = tail;
        newNode->next = *head;
        (*head)->prev = newNode;
    }
}
```

```

// Function to print the list
void printList(Node* head) {
    if (head == NULL) return;
    Node* temp = head;
    do {
        printf("address:%p %d\n", (void*)temp, temp->data);
        temp = temp->next;
    } while (temp != head);
}

```

```

void swapNodes(Node** head, Node* a, Node* b) {
    if (a == b || *head == NULL) return;

    Node* aPrev = a->prev;
    Node* aNext = a->next;
    Node* bPrev = b->prev;
    Node* bNext = b->next;

    if (*head == a) {
        *head = b;
    } else if (*head == b) {
        *head = a;
    }

    if (a->next == b) {
        a->next = bNext;
        bNext->prev = a;
        b->prev = aPrev;
        aPrev->next = b;
        b->next = a;
        a->prev = b;
    } else if (b->next == a) {
        b->next = aNext;
        aNext->prev = b;
        a->prev = bPrev;
        bPrev->next = a;
        a->next = b;
        b->prev = a;
    } else {
        aPrev->next = b;
        b->prev = aPrev;
    }
}

```

```

        aNext->prev = b;
        b->next = aNext;
        bPrev->next = a;
        a->prev = bPrev;
        bNext->prev = a;
        a->next = bNext;
    }
}

void sortList(Node** head) {
    if (*head == NULL) return;
    int swapped;
    Node* ptr1;
    Node* lptr = NULL;

    do {
        swapped = 0;
        ptr1 = *head;

        while (ptr1->next != lptr && ptr1->next != *head) {
            if (ptr1->data > ptr1->next->data) {
                swapNodes(head, ptr1, ptr1->next);
                swapped = 1;

                Node* temp = ptr1;
                ptr1 = ptr1->prev;
                ptr1->next = temp;
            }
            ptr1 = ptr1->next;
        }
        lptr = ptr1;
    } while (swapped);
}

int main() {
    int N;
    printf("Enter the number of elements: ");
    scanf("%d", &N);

    Node* head = NULL;
    for (int i = 0; i < N; i++) {
        int data;

```

```

        printf("Enter element %d: ", i + 1);
        scanf("%d", &data);
        insertEnd(&head, data);
    }

    printf("\nList before sorting:\n");
    printList(head);

    sortList(&head);

    printf("\nList after sorting:\n");
    printList(head);

    return 0;
}

```

## PENJELASAN

1. `#include <stdio.h>` - Menyertakan pustaka standar input-output untuk fungsi seperti `printf` dan `scanf`.
2. `#include <stdlib.h>` - Menyertakan pustaka standar untuk fungsi seperti `malloc` dan `free`.
- ;
- 3-7. `typedef struct Node { ... } Node;` - Mendefinisikan struktur `Node` yang berisi:
  - `int data;` - Menyimpan data integer.
  - `struct Node next;*` - Pointer ke node berikutnya dalam daftar.
  - `struct Node prev;*` - Pointer ke node sebelumnya dalam daftar.
8. `Node createNode(int data)*` - Mendefinisikan fungsi `createNode` yang membuat node baru.
9. `Node newNode = (Node)malloc(sizeof(Node));**` - Mengalokasikan memori untuk node baru dan mengembalikan pointer ke node tersebut.
10. `newNode->data = data;` - Menginisialisasi anggota data dari node baru dengan nilai yang diberikan.
11. `newNode->next = newNode->prev = newNode;` - Menginisialisasi pointer `next` dan `prev` dari node baru menunjuk ke node itu sendiri, membuatnya melingkar.
12. `return newNode;` - Mengembalikan pointer ke node baru.

13. `void insertEnd(Node head, int data)**` - Mendefinisikan fungsi `insertEnd` untuk menambahkan node baru di akhir daftar.
14. `Node newNode = createNode(data);*` - Membuat node baru dengan data yang diberikan.
15. `*if (head == NULL) {` - Mengecek apakah daftar kosong.
16. `*head = newNode;` - Jika kosong, node baru menjadi head.
17. `}` else { - Jika tidak kosong:
18. `Node tail = (head)->prev;` - Mendapatkan node terakhir (tail).
19. `tail->next = newNode;` - Menghubungkan node terakhir ke node baru.
20. `newNode->prev = tail;` - Menghubungkan node baru ke node terakhir.
21. `*newNode->next = head;` - Menghubungkan node baru ke node head.
22. `*(head)->prev = newNode;` - Menghubungkan node head ke node baru.
23. `}` - Akhir dari blok else.
24. `}` - Akhir dari fungsi `insertEnd`.
25. `void printList(Node head)*` - Mendefinisikan fungsi `printList` untuk mencetak elemen-elemen dalam daftar.
26. `if (head == NULL) return;` - Jika daftar kosong, keluar dari fungsi.
27. `Node temp = head;*` - Menyiapkan pointer sementara `temp` yang menunjuk ke head.
28. `do {` - Memulai loop do-while untuk mencetak elemen-elemen dalam daftar:
29. `printf("address:%p %d\n", (void)temp, temp->data);*` - Mencetak alamat dan data dari node saat ini.
30. `temp = temp->next;` - Berpindah ke node berikutnya.
31. `}` while (`temp != head`); - Ulangi sampai kembali ke head.
32. `}` - Akhir dari fungsi `printList`.
33. `void swapNodes(Node head, Node* a, Node* b)**` - Mendefinisikan fungsi `swapNodes` untuk menukar dua node dalam daftar.
34. `*if (a == b || head == NULL) return;` - Jika node yang akan ditukar sama atau daftar kosong, keluar dari fungsi.
35. `Node aPrev = a->prev;*` - Menyimpan pointer ke node sebelum `a`.
36. `Node aNext = a->next;*` - Menyimpan pointer ke node setelah `a`.

37. *Node bPrev = b->prev;*\* - Menyimpan pointer ke node sebelum b.

38. *Node bNext = b->next;*\* - Menyimpan pointer ke node setelah b.

39. *\*if (head == a) {* - Jika a adalah head, set head ke b.

40. *\*head = b;* - Set head ke b.

41. *\*} else if (head == b) {* - Jika b adalah head, set head ke a.

42. *\*head = a;* - Set head ke a.

43. *}* - Akhir dari blok else if.

44. *if (a->next == b) {* - Jika a dan b berdekatan, tukar a dan b dengan cara khusus:

45. *a->next = bNext;* - Set a->next ke node setelah b.

46. *bNext->prev = a;* - Set bNext->prev ke a.

47. *b->prev = aPrev;* - Set b->prev ke node sebelum a.

48. *aPrev->next = b;* - Set aPrev->next ke b.

49. *b->next = a;* - Set b->next ke a.

50. *a->prev = b;* - Set a->prev ke b.

51. *}* *else if (b->next == a) {* - Jika b dan a berdekatan (dalam urutan terbalik):

52. *b->next = aNext;* - Set b->next ke node setelah a.

53. *aNext->prev = b;* - Set aNext->prev ke b.

54. *a->prev = bPrev;* - Set a->prev ke node sebelum b.

55. *bPrev->next = a;* - Set bPrev->next ke a.

56. *a->next = b;* - Set a->next ke b.

57. *b->prev = a;* - Set b->prev ke a.

58. *}* *else {* - Jika a dan b tidak berdekatan:

59. *aPrev->next = b;* - Set aPrev->next ke b.

60. *b->prev = aPrev;* - Set b->prev ke aPrev.

61. *aNext->prev = b;* - Set aNext->prev ke b.

62. *b->next = aNext;* - Set b->next ke aNext.

63. *bPrev->next = a;* - Set bPrev->next ke a.

64. `a->prev = bPrev;` - Set `a->prev` ke `bPrev`.

65. `bNext->prev = a;` - Set `bNext->prev` ke `a`.

66. `a->next = bNext;` - Set `a->next` ke `bNext`.

67. `}` - Akhir dari blok `else`.

68. `}` - Akhir dari fungsi `swapNodes`.

69. `void sortList(Node head)**` - Mendefinisikan fungsi `sortList` untuk mengurutkan daftar menggunakan algoritma bubble sort.

70. `*if (head == NULL) return;` - Jika daftar kosong, keluar dari fungsi.

71. `int swapped;` - Mendeklarasikan variabel `swapped` untuk melacak apakah ada pertukaran yang dilakukan.

72. `Node ptr1;*` - Mendeklarasikan pointer `ptr1` untuk iterasi melalui daftar.

73. `Node lptr = NULL;*` - Mendeklarasikan pointer `lptr` untuk menandai akhir dari bagian yang sudah diurutkan.

74. `do {` - Memulai loop `do-while` untuk mengulang proses pengurutan:

75. `swapped = 0;` - Menginisialisasi `swapped` ke 0 pada awal setiap iterasi.

76. `*ptr1 = head;` - Set `ptr1` ke `head`.

77. `*while (ptr1->next != lptr && ptr1->next != head) {` - Loop melalui daftar hingga mencapai bagian yang sudah diurutkan atau kembali ke `head`:

78. `if (ptr1->data > ptr1->next->data) {` - Jika data di node `ptr1` lebih besar dari data di node berikutnya:

79. `swapNodes(head, ptr1, ptr1->next);` - Tukar `ptr1` dan node berikutnya.

80. `swapped = 1;` - Set `swapped` ke 1, menandakan bahwa ada pertukaran yang dilakukan.

81. `Node temp = ptr1;*` - Menyimpan `ptr1` dalam variabel sementara `temp`.

82. `ptr1 = ptr1->prev;` - Set `ptr1` ke node sebelumnya.

83. `ptr1->next = temp;` - Memperbarui pointer `next` dari node sebelumnya untuk menunjuk ke node sementara `temp`.

84. `}` - Akhir dari blok `if`.

85. `ptr1 = ptr1->next;` - Berpindah ke node berikutnya.

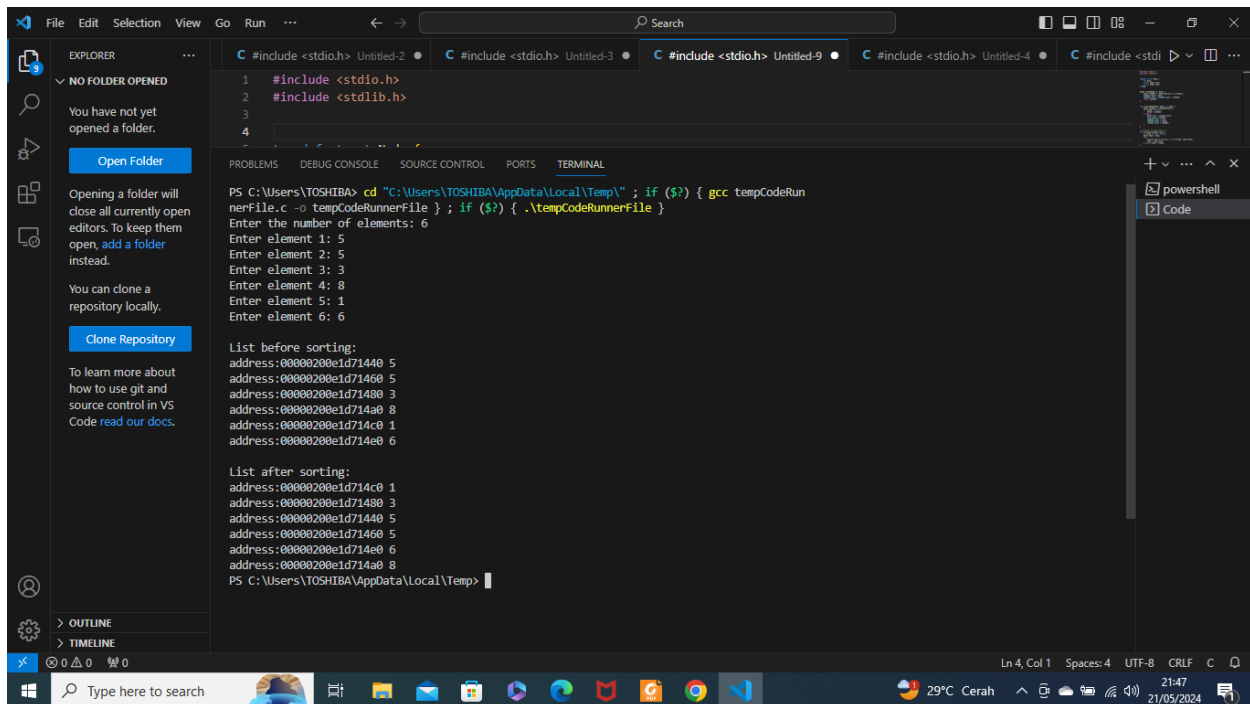
86. `}` - Akhir dari loop `while`.

87. `lptr = ptr1;` - Memperbarui `lptr` ke `ptr1`, menandai akhir dari bagian yang sudah diurutkan.

88. } while (swapped); - Ulangi proses jika ada pertukaran yang dilakukan.
89. } - Akhir dari fungsi sortList.
90. int main() - Fungsi utama program.
91. int N; - Mendeklarasikan variabel N untuk menyimpan jumlah elemen yang akan dimasukkan.
92. printf("Enter the number of elements: "); - Meminta pengguna untuk memasukkan jumlah elemen.
93. scanf("%d", &N); - Membaca jumlah elemen dari input pengguna.
94. *Node head = NULL;*\* - Mendeklarasikan pointer head dan menginisiasinya ke NULL.
95. for (int i = 0; i < N; i++) { - Loop untuk menginput elemen-elemen ke dalam daftar:
96. int data; - Mendeklarasikan variabel data untuk menyimpan nilai input.
97. printf("Enter element %d: ", i + 1); - Meminta pengguna untuk memasukkan elemen ke-i.
98. scanf("%d", &data); - Membaca nilai elemen dari input pengguna.
99. insertEnd(&head, data); - Menambahkan elemen ke akhir daftar.
100. } - Akhir dari loop for.
101. printf("\nList before sorting:\n"); - Mencetak pesan sebelum menampilkan daftar yang belum diurutkan.
102. printList(head); - Memanggil printList untuk mencetak daftar yang belum diurutkan.
103. sortList(&head); - Memanggil sortList untuk mengurutkan daftar.
104. printf("\nList after sorting:\n"); - Mencetak pesan sebelum menampilkan daftar yang sudah diurutkan.
105. printList(head); - Memanggil printList untuk mencetak daftar yang sudah diurutkan.
106. return 0; - Mengakhiri program dengan kode status 0 (sukses).



## OUTPUT



The screenshot shows the Visual Studio Code interface with a C program in a file named 'Untitled-9'. The program includes `<stdio.h>` and `<stdlib.h>`. It prompts the user to enter 6 numbers. The output in the terminal shows the addresses of the numbers before and after sorting. The addresses are: 00000200e1d71440, 00000200e1d71460, 00000200e1d71480, 00000200e1d714a0, 00000200e1d714c0, and 00000200e1d714e0. The addresses are sorted in ascending order.

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

int main()
{
    int arr[6];
    for (int i = 0; i < 6; i++)
    {
        printf("Enter element %d: ", i + 1);
        scanf("%d", &arr[i]);
    }

    printf("\nList before sorting:\n");
    for (int i = 0; i < 6; i++)
    {
        printf("address: %p %d\n", &arr[i], arr[i]);
    }

    qsort(arr, 6, sizeof(int), (int (*)(const void *, const void *)) NULL);

    printf("\nList after sorting:\n");
    for (int i = 0; i < 6; i++)
    {
        printf("address: %p %d\n", &arr[i], arr[i]);
    }

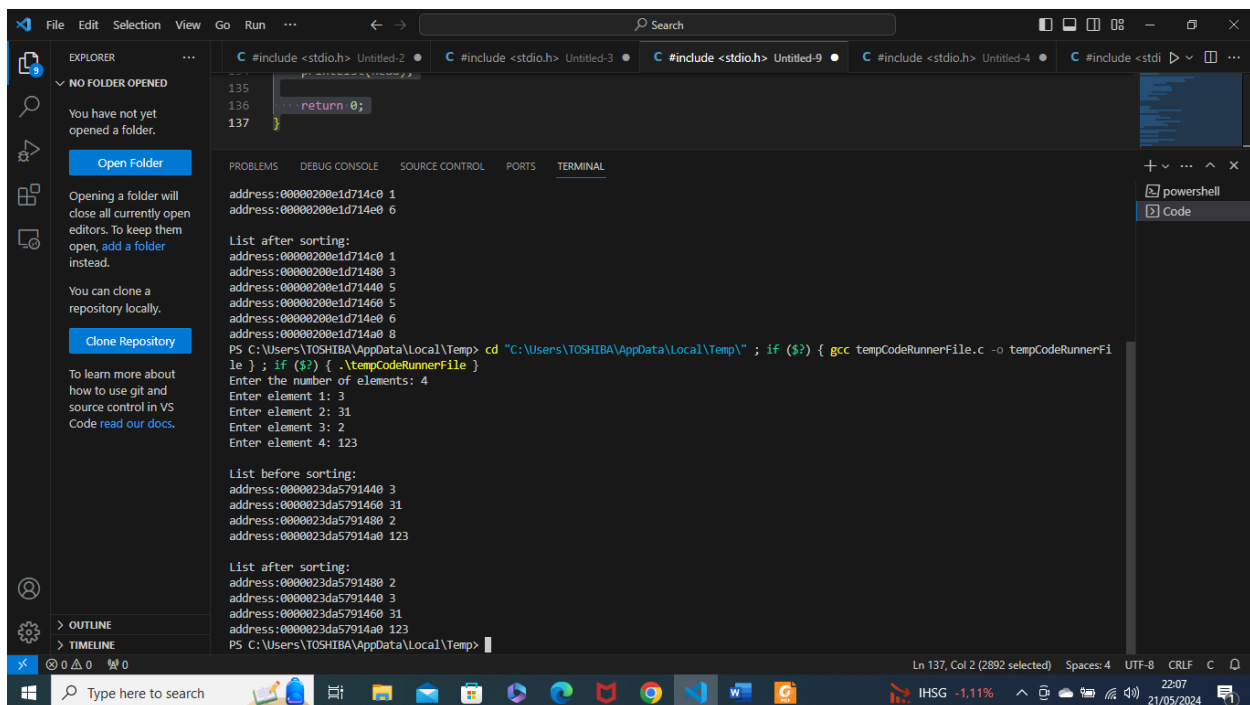
    return 0;
}
```

PS C:\Users\TOSHIBA\AppData\Local\Temp\> cd "C:\Users\TOSHIBA\AppData\Local\Temp\" ; if (\$?) { gcc tempCodeRunnerFile.c -o tempCodeRunnerFile ; if (\$?) { .\tempCodeRunnerFile } }

Enter the number of elements: 6  
Enter element 1: 5  
Enter element 2: 5  
Enter element 3: 3  
Enter element 4: 8  
Enter element 5: 1  
Enter element 6: 6

List before sorting:  
address:00000200e1d71440 5  
address:00000200e1d71460 5  
address:00000200e1d71480 3  
address:00000200e1d714a0 8  
address:00000200e1d714c0 1  
address:00000200e1d714e0 6

List after sorting:  
address:00000200e1d714c0 1  
address:00000200e1d71480 3  
address:00000200e1d71440 5  
address:00000200e1d71460 5  
address:00000200e1d714e0 6  
address:00000200e1d714a0 8  
PS C:\Users\TOSHIBA\AppData\Local\Temp\>



The screenshot shows the Visual Studio Code interface with a C program in a file named 'Untitled-9'. The program includes `<stdio.h>` and `<stdlib.h>`. It prompts the user to enter 4 numbers. The output in the terminal shows the addresses of the numbers before and after sorting. The addresses are: 0000023da5791440, 0000023da5791460, 0000023da5791480, and 0000023da57914a0. The addresses are sorted in ascending order.

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

int main()
{
    int arr[4];
    for (int i = 0; i < 4; i++)
    {
        printf("Enter element %d: ", i + 1);
        scanf("%d", &arr[i]);
    }

    printf("\nList before sorting:\n");
    for (int i = 0; i < 4; i++)
    {
        printf("address: %p %d\n", &arr[i], arr[i]);
    }

    qsort(arr, 4, sizeof(int), (int (*)(const void *, const void *)) NULL);

    printf("\nList after sorting:\n");
    for (int i = 0; i < 4; i++)
    {
        printf("address: %p %d\n", &arr[i], arr[i]);
    }

    return 0;
}
```

PS C:\Users\TOSHIBA\AppData\Local\Temp\> cd "C:\Users\TOSHIBA\AppData\Local\Temp\" ; if (\$?) { gcc tempCodeRunnerFile.c -o tempCodeRunnerFile ; if (\$?) { .\tempCodeRunnerFile } }

Enter the number of elements: 4  
Enter element 1: 3  
Enter element 2: 31  
Enter element 3: 2  
Enter element 4: 123

List before sorting:  
address:0000023da5791440 3  
address:0000023da5791460 31  
address:0000023da5791480 2  
address:0000023da57914a0 123

List after sorting:  
address:0000023da5791480 2  
address:0000023da5791440 3  
address:0000023da5791460 31  
address:0000023da57914a0 123  
PS C:\Users\TOSHIBA\AppData\Local\Temp\>

