สัปดาห์ที่ 8 Linked Lists

เรียบเรียงโดย ชาคริต วัชโรภาส

วิชา 01418113 Computer Programming

1. Linked List คืออะไร

- Linked list เป็นโครงสร้างข้อมูลที่ถูกออกแบบมาใช้เก็บข้อมูล โดยที่ข้อมูลแต่ละตัวจะเชื่อมโยงต่อกันเป็นสาย เพื่อผู้ใช้สามารถเข้าถึงข้อมูลแต่ละตัวได้ (ตราบที่สายข้อมูลยังไม่ขาดออกจากกัน)
- Array ก็เป็นโครงสร้างข้อมูลที่มีลักษณะที่ข้อมูลแต่ละตัวอยู่เรียงต่อกัน แต่ทั้ง array และ linked list มีโครงสร้างและการจัดการในการเก็บข้อมูลที่ไม่เหมือนกันซะทีเดียว

2. เกริ่นนำเกี่ยวกับ Structure

- Structure เป็นการกำหนดรูปแบบหรือโครงสร้างข้อมูลที่มีการรวบรวมตัวแปรที่เก็บข้อมูลหลายเรื่องที่สัมพันธ์กัน ที่อาจมีประเภทข้อมูลที่แตกต่างเข้าไว้ด้วยกัน
- ตัวอย่างการกำหนด structure

```
struct student {
   char id[12];
   char name[40];
   int age;
};
```

2.1 การประกาศตัวแปรประเภท Structure

- เราสามารถประกาศตัวแปรที่มีโครงสร้าง structure ได้ดังนี้

```
struct student std;
```

โดยที่ std เป็นตัวแปรที่มีโครงสร้างตาม struct student

- การเข้าถึงสมาชิกแต่ละฟิลด์ของ structure สามารถทำได้โดยใช้ **dot operator** คือใช้เครื่องหมาย . ดังตัวอย่างเช่น

```
printf("%s\n", std.name);
std.age = 21;
```

2.2 การประกาศตัวแปรพร้อมกำหนดค่าเริ่มต้นให้กับตัวแปรประเภท Structure

- เราสามารถประกาศพร้อมกำหนดค่าเริ่มต้นให้กับตัวแปรที่มีโครงสร้าง structure ได้ดังนี้

```
struct student another_std = {"6211140123", "Josh Smith", 18};
```

2.3 ตัวแปร pointer ไปยัง Structure

- การเข้าถึงสมาชิกแต่ละฟิลด์ผ่านตัวแปร pointer สามารถทำได้ในลักษณะนี้

```
struct student std, *p_std;

p_std = &std;
printf("%s\n", (*p_std).name);
(*p_std).age = 21;
```

- นอกจากการใช้ dereferencing operator กับตัวแปร pointer เพื่อเข้าถึงฟิลด์ผ่าน . (dot operator) แล้ว เรายังสามารถใช้ -> (structure pointer operator) หรือเรียกอีกอย่างหนึ่งว่า **arrow operator**

2.4 การใช้ typedef

- เราสามารถกำหนดประเภทข้อมูลขึ้นมาเพิ่มเติมผ่านการใช้คีย์เวิร์ด typedef ได้
- ตัวอย่างการใช้งาน

```
typedef unsigned long int ulint;
```

- สำหรับกรณีของ struct ก็เช่นกัน เราสามารถกำหนด struct student ให้กลายเป็นประเภทข้อมูลใหม่ที่ใช้เพียง 1 คำแทนประเภทข้อมูลได้

```
typedef struct student {
   char id[12];
   char name[40];
   int age;
} Student;
```

หรือ

```
typedef struct {
   char id[12];
   char name[40];
   int age;
} Student;
```

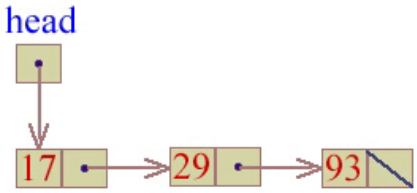
2.5 Self-Referential Structures

- Self-referential structure เป็นโครงสร้างที่เราออกแบบมาเพื่อจัดการ linked list โดยเราสร้างขึ้นมาด้วยการใช้ struct และภายใน struct นี้จะมีฟิลด์ที่เป็น pointer เพื่อใช้อ้างอิงกลับไปยัง struct นี้อีก
- Struct นี้ถึงได้ถูกเรียกว่า self-referential structure ซึ่งก็คือ struct ที่อ้างกลับไปยังโครงสร้างตนเอง
- ตัวอย่างโครงสร้าง struct

```
struct node {
     int data;
     struct node *next;
};
```

- เรามักจะใช้คำว่า "node" ในการอ้างถึงหน่วยหรือโหนดที่ออกแบบมาเพื่อเก็บข้อมูล 1 หน่วย
- พูดง่ายๆ ก็คือ ถ้าเราต้องการเก็บข้อมูล 10 ตัว เราก็จะใช้โหนด 10 โหนดในการเก็บข้อมูลเหล่านี้
- struct node นี้จะเก็บข้อมูล 2 ส่วนคือ
  - ส่วนที่จะใช้เก็บข้อมูล ซึ่งในตัวอย่างนี้ เราใช้ฟิลด์ data ในการเก็บข้อมูลประเภท int
  - อีกส่วนจะเป็นตัวแปร pointer ที่ใช้อ้างอิงไปยังโหนดถัดไปใน linked list ซึ่งในตัวอย่างนี้ เราใช้ฟิลด์ next ในการเก็บตำแหน่งในหน่วยความจำของโหนดถัดไป

3. ตัวอย่าง linked list ที่มีข้อมูล



- หากเราต้องการใช้โครงสร้างโหนดด้านบนมาใช้เพื่อจัดเก็บข้อมูล 3 โหนดตามรูปนี้ เราสามารถเขียนโค้ดออกมาได้อย่างไร

ลองดูโค้ดข้างล่างนี้

In [*]:

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

struct node {
   int data;
   struct node *next;
};

int main()
{  struct node *head = NULL;
   struct node *first = NULL;
   struct node *second = NULL;
   struct node *third = NULL;

   // Allocate 3 nodes in the heap
   first = (struct node *)malloc(sizeof(struct node));
   second = (struct node *)malloc(sizeof(struct node));
   third = (struct node *)malloc(sizeof(struct node));

   head = first;
   first->data = 17;
   first->next = second;

   second->data = 29;
   second->next = third;

   third->data = 93;
   third->next = NULL;
}
```

4. การเข้าถึงข้อมูลแต่ละตัวใน linked list (Traversal)

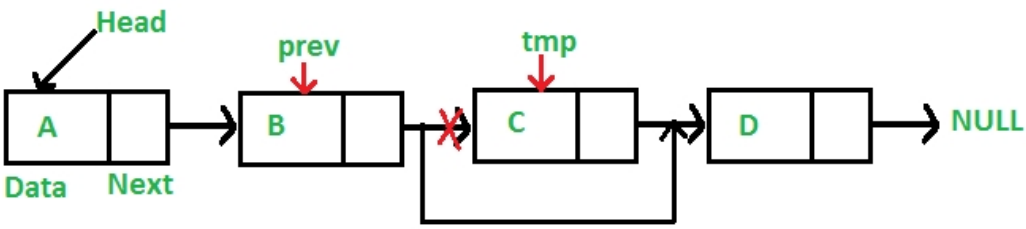
- เราจะปรับโค้ดก่อนหน้าโดยให้มีฟังก์ชันที่ใช้สร้าง linked list ที่มี 3 โหนด (ตามที่เคยทำไว้) แล้วเราจะสร้างอีกฟังก์ชันขึ้นมาเพื่อเข้าถึงข้อมูลโดยการแสดงข้อมูลแต่ละตัวออกมาบนหน้าจอ

In [*]:

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

struct node {
   int data;
   struct node *next;
};

struct node *create_list()
{  struct node *first, *second, *third;

   first = (struct node *)malloc(sizeof(struct node));
   second = (struct node *)malloc(sizeof(struct node));
   third = (struct node *)malloc(sizeof(struct node));

   first->data = 17;
   first->next = second;

   second->data = 29;
   second->next = third;

   third->data = 93;
   third->next = NULL;

   return first;
}

void print_list(struct node *head)
{  struct node *tmp;

   for (tmp=head; tmp; tmp = tmp->next)
      printf("%d\n", tmp->data);
}

int main()
{  struct node *head;

   head = create_list();
   print_list(head);
}
```

5. การลบข้อมูลออกจาก linked list (Deletion)

- ภาพจาก [geeksforgeeks.org (https://www.geeksforgeeks.org/linked-list-set-3-deleting-node)](https://www.geeksforgeeks.org/linked-list-set-3-deleting-node)



- หากเราต้องการลบข้อมูลออกไปจาก list เราจะต้องจัดการกับ linked list นี้อย่างไร
- เราจะลองเขียนฟังก์ชันเพื่อใช้ลบข้อมูลที่ต้องการออกจาก list

In [*]:

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

struct node {
   int data;
   struct node *next;
};

struct node *create_list()
{  struct node *first, *second, *third;

   first = (struct node *)malloc(sizeof(struct node));
   second = (struct node *)malloc(sizeof(struct node));
   third = (struct node *)malloc(sizeof(struct node));

   first->data = 17;
   first->next = second;

   second->data = 29;
   second->next = third;

   third->data = 93;
   third->next = NULL;

   return first;
}

void print_list(struct node *head)
{  struct node *tmp;

   for (tmp=head; tmp; tmp = tmp->next)
      printf("%d\n", tmp->data);
}

void delete_node(struct node **head_ref, int key)
{  struct node* tmp = *head_ref, *prev;

   if (tmp != NULL && tmp->data == key) {
      *head_ref = tmp->next;
      free(tmp);
      return;
   }

   while (tmp != NULL && tmp->data != key) {
      prev = tmp;
      tmp = tmp->next;
   }

   if (tmp == NULL) return;

   prev->next = tmp->next;

   free(tmp);
}

int main()
{  struct node *head;
```

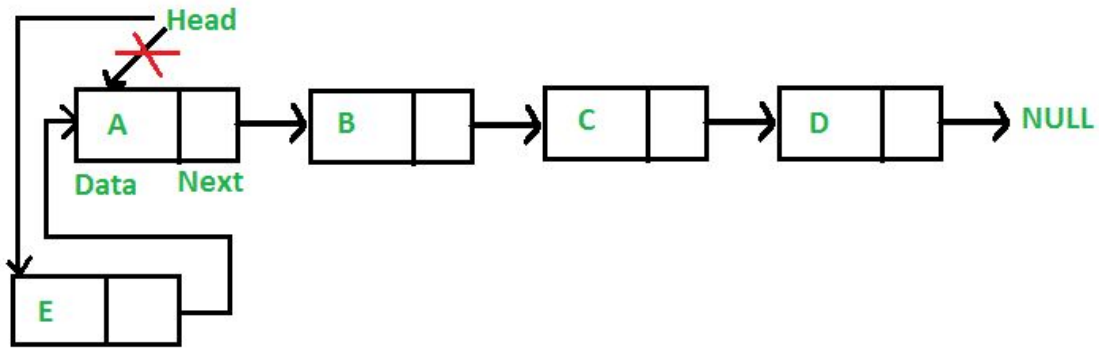
```
    int value;

    head = create_list();
    printf("Before: \n");
    print_list(head);
    value = 17;
    printf("After deleting %d: \n", value);
    delete_node(&head, value);
    print_list(head);
}
```

6. การเพิ่มข้อมูลเข้าไปใน linked list (Insertion)

- การเพิ่มโหนดเข้าไปใน list สามารถทำได้ในหลายลักษณะ โดยพิจารณาจากตำแหน่งใน list ที่ต้องการเพิ่มโหนดเข้าไป

การเพิ่มเข้าไปในส่วนต้นของ list

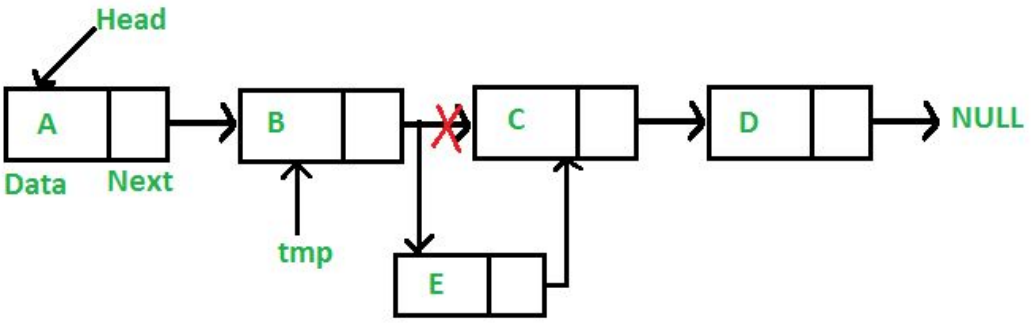
- ภาพจาก geeksforgeeks.org (https://www.geeksforgeeks.org/linked-list-set-2-inserting-a-node/)




```
void insert_node_at_front(struct node **head_ref, int new_data)
{
    struct node *new_node = (struct node *)malloc(sizeof(struct node));
    new_node->data = new_data;

    new_node->next = *head_ref;
    *head_ref = new_node;
}

int main()
{   struct node *head;
    int value;

    head = create_list();
    printf("Before: \n");
    print_list(head);
    value = 5;
    printf("After inserting %d: \n", value);
    insert_node_at_front(&head, value);
    print_list(head);
}
```

การเพิ่มเข้าไปในตอนกลางของ list

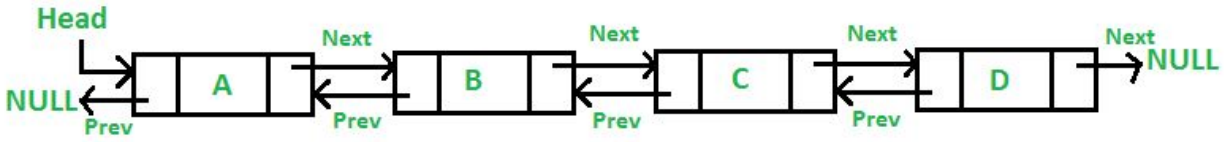
- ภาพจาก [geeksforgeeks.org (https://www.geeksforgeeks.org/linked-list-set-2-inserting-a-node/)](https://www.geeksforgeeks.org/linked-list-set-2-inserting-a-node/)




```
void insert_node_after(struct node *prev_node, int new_data)
{
   struct node *new_node = (struct node *)malloc(sizeof(struct node));
   new_node->data = new_data;

   new_node->next = prev_node->next;
   prev_node->next = new_node;
}

int main()
{  struct node *head, *tmp;
   int search_value, insert_value;

   head = create_list();
   printf("Before: \n");
   print_list(head);
   search_value = 29;
   for (tmp=head; tmp; tmp=tmp->next)
      if (tmp->data == search_value)
         break;
   if (tmp) {
      insert_value = 50;
      printf("After inserting %d after %d:\n", insert_value, search_value);
      insert_node_after(tmp, insert_value);
   }
   print_list(head);
}
```

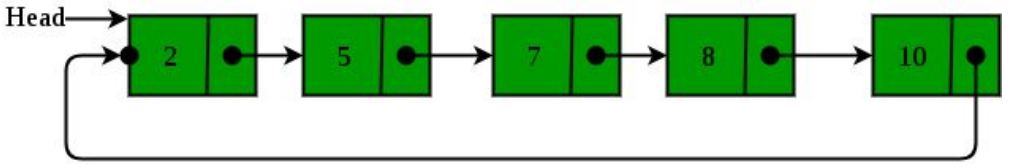
7. Doubly Linked List

- ลักษณะของ linked list ที่เห็นผ่านมาเรียกว่า Singly Linked List โดยพิจารณาได้จากการที่แต่ละโหนดที่อยู่ติดกันเชื่อมต่อถึงกันผ่านลิงก์เพียงลิงก์เดียว
- นอกจาก Singly Linked List เรายังสามารถสร้างโครงสร้างข้อมูลที่เรียกว่า Doubly Linked List โดยที่แต่ละโหนดที่อยู่ติดกันเชื่อมต่อกันด้วยลิงก์ 2 ลิงก์
- ลิงก์อันนึงใช้เชื่อมต่อไปยังโหนดถัดไป และลิงก์อีกอันนึงใช้เชื่อมต่อไปยังโหนดก่อนหน้า ดังแสดงในรูป

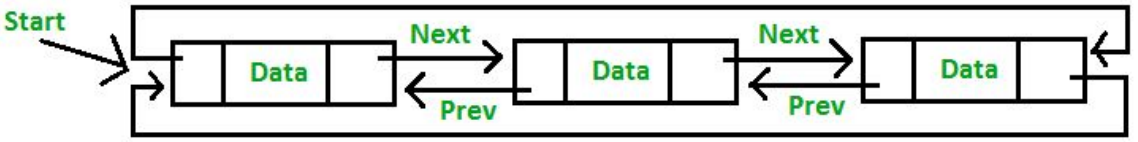


8. Circular Linked List

- Singly Circular Linked List



- Doubly Circular Linked List



- ภาพจาก geeksforgeeks.org (https://www.geeksforgeeks.org/doubly-linked-list/)