CP 111 Computer Programming (2/2554 Section B01-B02)

Handout 12

<u>หัวข้อ:</u>

Data structures (Chapter 19 textbook)

- ทบทวน: ตัวแปรแบบ structures ในภาษา C, Arrays of structures และ Pointers and Structures, Dynamic Memory Allocation: Dynamically sized array of structures
- Linked lists

<u>ทบทวน</u>

1. ตัวแปรที่มีชนิดข้อมูล (derived data type) แบบ structures → จัดรวมข้อมูลที่มี data type ต่างกันมาอยู่ด้วยกัน
2. สรุป รูปแบบ (syntax) ภาษา C ที่เกี่ยวข้องกับ structures

(2.1) นิยามโครงสร้างของ structures (นิยามชื่อและชนิดข้อมูลฟิลด์ย่อย)	`typedef struct`
	`{`
	`  <type> <field_name_1>;`
	`  <type> <field_name_2>;`
	`  …`
	`} TYPE_NAME;`
	
	หรือ
	`struct tag_name`
	`{`
	`  <type> <field_name_1>;`
	`  <type> <field_name_2>;`
	`  …`
	`}`
	`typedef struct tag_name TYPE_NAME;`
		<u>ตัวอย่าง</u>
		`#include <stdio.h>`
		`typedef struct`
		`{`
		`  char name[10];`
		`  int ID;`
		`  double grade;`
		`} S_INFO;`
		
		`/*`
		`struct s_info_tag {`
		`  char name[10];`
		`  int ID;`
		`  double grade;`
		`}`
		`typedef struct s_info_tag S_INFO;`
		`*/`
		`void main()`
		`{`
		`S_INFO st1;`
		`S_INFO s[100];`
		`S_INFO *ptr;`
		
		`  gets(st1.name);`
		`  st1.ID=55009999;`
		
		`  gets(s[0].name);`
		`  s[0].grade=4.0;`
		
		`  ptr=&st1;`
		`  gets(ptr->name);`
		`  (*ptr).ID=55009999;`
		`  ptr->ID=55009999;`
		`}`

(2.2) ประกาศตัวแปร	`TYPE_NAME <var_name>;`	
(2.3) การอ้างถึงแต่ละฟิลด์ย่อย ในตัวแปรแบบ structure	`<var_name>.<field_name>`	

(2.4) การอ้างถึงแต่ละฟิลด์ย่อย ในตัวแปรแบบ structure ผ่านตัวแปรแบบ pointer	`(*<pointer_name>).<field_name>`
	
	หรือ
	
	`<pointer_name>-><field_name>`	

3. โจทย์ตัวอย่าง: จงเขียนโปรแกรมเพื่อทำการหาค่าผลคูณ dot product ของเวค์เตอร์ในปริภูมิ 3 มิติ (3D Vector) และหาเวคเตอร์ผลลัพธ์ที่เกิดจาก 2 เวคเตอร์มาบวกกัน เช่น ถ้า V1=[1 2 3] และ V2=[-3 1 1] แล้วค่า dot product ของ V1 และ V2 จะเท่ากับ 1*(-3) + 2*1 + 3*1 = 1 และ เวคเตอร์ผลลัพธ์จากการบวกคือ V3=V1+V2=[-2 3 4]

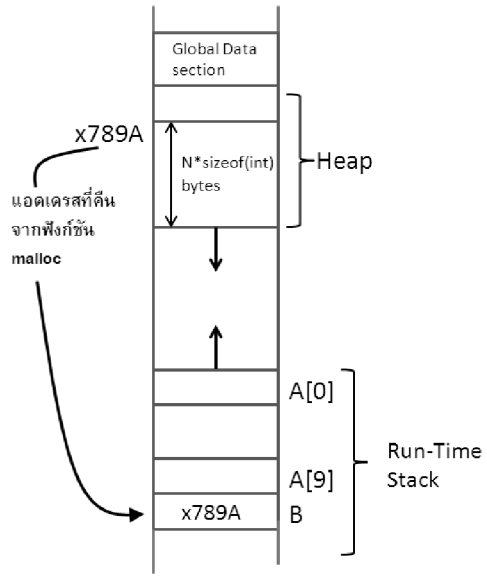
กำหนดให้ใช้ตัวแปรแบบ structure ที่มีชนิดข้อมูลชื่อ VEC3 ซึ่งประกอบด้วยฟิลด์ย่อย vx, vy และ vz สำหรับเก็บส่วนประกอบ (component)ในแกน x, y และ z ตามลำดับ

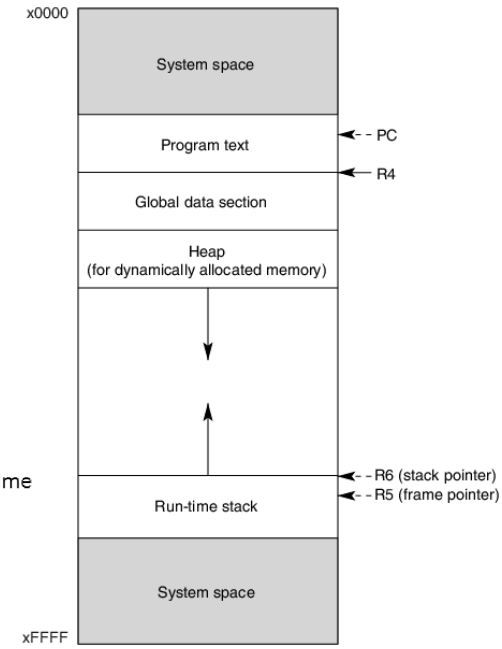
```
#include <stdio.h>



..................................................... // ใช้ typedef นิยามโครงสร้างของ VEC3
#define V_DOT(u,v) ..................................................
...................V_add(.....................................................);
int main()
{
VEC3 v1, v2, v3, *ptr;
 double dp_val;
   scanf("%lf %lf %lf",.....................................................); // อ่าน v1 จาก Keyboard
   scanf("%lf %lf %lf",.....................................................); // อ่าน v2 จาก Keyboard
   ..................................................... // เรียก macro V_DOT
   .....................................................// เรียกฟังก์ชัน Vadd
   printf("%.2lf\n",dp_val);
   ptr=&v3;
   printf("%.2f %.2f %.2f\n",.....................................................);
}

.....................................................// โค้ดฟังก์ชัน V_add
{




}
```

4. Dynamic memory allocation: ตัวแปรหนึ่งๆในโปรแกภาษา C จะถูกจัดให้เก็บในหน่วยความจำที่ใดที่หนึ่งใน 3 บริเวณต่อไปนี้ โดยที่ตัวอย่างของ Memory map ในกรณีของคอมพิวเตอร์ LC-3 แสดงดังรูปข้างล่าง

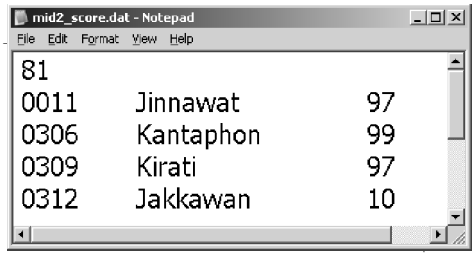
- Run-time stack: สำหรับเก็บตัวแปรแบบ local variables ต่างๆ ที่ประกาศในแต่ละฟังก์ชัน
- Global data section: สำหรับเก็บตัวแปรแบบ global variables
- Heap: สำหรับเก็บเนื้อที่ที่ถูกสร้างในขณะ run-time เช่น dynamically sized array ที่ได้เรียนมาแล้ว

```
int g_C[10];
void main()
{
int A[10];
int *B;
int i,N;
       scanf("%d",&N);
       B=(int *)malloc(N*sizeof(int));
       for(i=0;i<N;i++)
              scanf("%s",&B[i]);
       ...........
       free(B);
}
```





5. โจทย์ตัวอย่าง: จงเขียนโปรแกรมเพื่อแสดงข้อมูลของนักเรียนที่ได้คะแนนน้อยสุดที่มากกว่าคะแนนเฉลี่ยของนักเรียนทั้งหมด กำหนดให้ข้อมูลนำเข้าเก็บอยู่ในไฟล์ d:\cp111\mid2_score.dat แสดงดังรูปข้างล่าง ซึ่งบรรทัดแรกจะแสดงจำนวนนักเรียน สมมติคือ N หลังจากนั้น N บรรทัดถัดมาจะเป็นข้อมูลคะแนนสอบของนักเรียนแต่ละคน ซึ่งประกอบด้วย รหัส ชื่อ และคะแนนสอบ กำหนดให้การเขียนโปรแกรมให้สร้าง Dynamically-sized array of structures โดยให้นิยาม structure ที่เหมาะสม




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




.....................................................// ใช้ typedef นิยามโครงสร้าง
void main()
{
.....................................................// ประกาศตัวแปร pointer to structure
FILE *infile;
int i,N,sum,min;
double avg;
              ..............................................................
             ..............................................................
            {
                printf("cannot open file\n");
                return;
                       }
           ..............................................................// อ่านจำนวนนักเรียนเก็บไว้ที่ N
                                                                          // จองเนื้อที่ในหน่วยความจำ
             ..............................................................
            {
                printf("cannot allocate memory\n");
                return;
            }
            sum=0;  // วนอ่านข้อมูลเก็บในอาร์เรย์ และคำนวณหาค่าเฉลี่ย
            for(i=0;i<N;i++)
            {
              ..............................................................
             ..............................................................
            }
             ..............................................................  // อ่านข้อมูลครบ -> ปิดไฟล์
            avg=(double)sum/N;
            min=100;
            min_i=-1;
            for(i=0;i<N;i++)
            {
              ..............................................................
              ..............................................................
              ..............................................................
            }
             printf("%d %s %d\n",..............................................);
.
            ..............................................................  // คืนเนื้อที่ในหน่วยความจำ
}
```

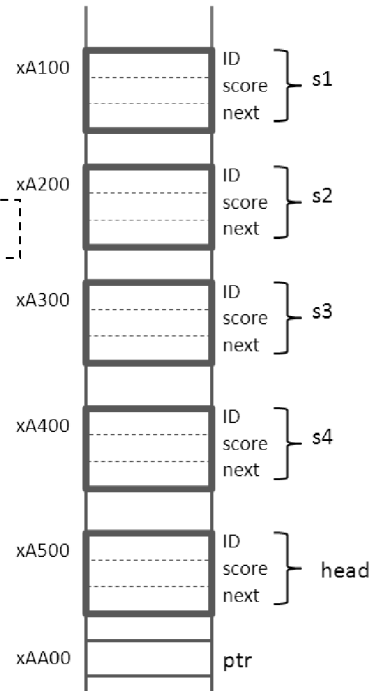
Linked lists

1. พิจารณาโปรแกรมตัวอย่างข้างล่าง ซึ่งแสดงตัวอย่างการสร้างโครงสร้างข้อมูลแบบ Linked list แบบง่าย

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

typedef struct s_info_tag S_INFO_NODE;
struct s_info_tag {
  int ID;
  int score;
  S_INFO_NODE *next;
};

void main()
{
S_INFO_NODE s1,s2,s3,s4,head;
S_INFO_NODE *ptr;

     s1.ID=10001;
     s1.score=99;
     s2.ID=10012;
     s2.score=89;
     s3.ID=10025;
     s3.score=78;
     s4.ID=10089;
     s4.score=96;

     head.next=&s1;
     s1.next=&s2;
     s2.next=&s3;
     s3.next=&s4;
     s4.next=NULL;   // Line A

     ptr=head.next;
     while(ptr!=NULL)
     {
       // Line B
       printf("%d %d\n",ptr->ID,ptr->score);
       ptr=ptr->next;
     }
}
```

self-referential structures

Q1: สมมติว่าตัวแปรต่างๆ เก็บอยู่ในหน่วยความจำที่แอดเดรสต่างๆ ดังรูปข้างล่าง จงระบุค่าของตัวแปร เมื่อโปรแกรมทำงานถึง Line A



Q2: จงระบุค่าต่างๆ ของ ptr ที่เปลี่ยนไปเมื่อโปรแกรมทำงานที่ Line B พร้อมทั้งระบุผลลัพธ์ที่หน้าจอ

รอบที่ 1 ptr=.....แสดง.....

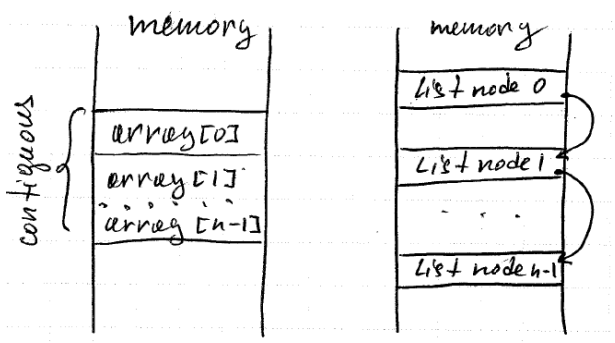
รอบที่ 1 ptr=.....แสดง.....

รอบที่ 2 ptr=.....แสดง.....

รอบที่ 4 ptr=.....แสดง.....

2. Linked list เป็นโครงสร้างข้อมูลพื้นฐานที่มีการใช้งานอย่างมากแบบหนึ่งในการเขียนโปรแกรม ซึ่งอธิบายได้ดังนี้

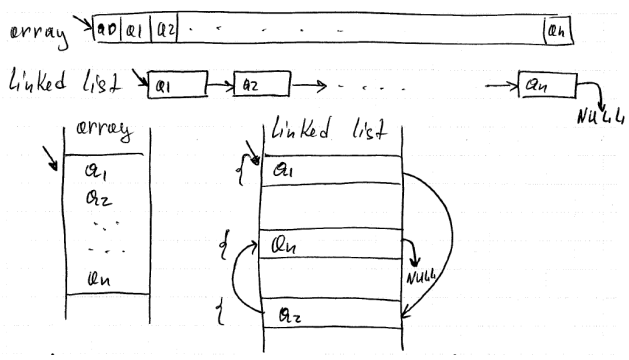
- มีลักษณะคล้ายกับ array ตรงที่จะใช้สำหรับชุดของข้อมูลหลายๆตัว (list of elements)
- มีลักษณะแตกต่างจาก array ตรงที่ แต่ละ element จะไม่ได้ถูกเก็บในหน่วยความจำที่ต่อกัน (contiguous memory)



รูปนำมาจาก slide วิชา ECE190 UIUC by V.Kindratenko (https://wiki.engr.illinois.edu/display/ece190/Lectures)

- อาร์เรย์จะมีขนาดคงที่ เมื่อมันถูกสร้างขึ้นมาได้จองเนื้อที่ในหน่วยความจำไว้
- การเพิ่ม element ใหม่ลงในอาร์เรย์และการลบ element ที่มีอยู่แล้วออกจากอาร์เรย์ จะค่อนข้างยุ่งยาก
- ตัวอย่าง: การลบ element หนึ่งๆ ออกจากอาร์เรย์ จะต้องจองเนื้อที่หน่วยความจำใหม่สำหรับอาร์เรย์ที่มีขนาดลดลงไป 1 ตัว หลังจากนั้นจะต้อง copy ข้อมูลในอาร์เรย์เก่าไปยังอาร์เรย์ใหม่ โดยไม่ copy ตัวที่จะลบออกไป

3. Linked list สามารถมองได้เป็นชุด (collection) ของ Node ซึ่งแต่ละ node จะเก็บข้อมูล 1 element โดยที่ linked list จะไม่เก็บข้อมูลแต่ละ element เรียงต่อกันในหน่วยความจำ แต่จะใช้การเก็บแต่ละ node แยกกัน โดยมี pointer ที่ชี้ node หนึ่งๆไปยัง node ถัดไป



- Linked list จะมี node ที่เป็น node แรก ซึ่งเก็บข้อมูล element แรก โดยที่ node แรกนี้จะเรียกว่า head และจะมี node ที่เป็น node สุดท้ายซึ่งจะเรียกว่า tail ซึ่ง pointer ของ node นี้จะชี้ไปที่ NULL
- <u>ข้อแตกต่างที่สำคัญระหว่าง array กับ linked list</u> คือ

(i) เราสามารถอ้างถึง element ใดๆ ใน array ก็ได้ ณ เวลาหนึ่งๆ โดยใช้ array index เช่น A[i]

(ii) แต่การอ้างถึง element ใดๆ ใน linked list จะต้องเริ่มจาก node แรกแล้ววิ่งไปที่ละ node จนกว่าจะถึง node ที่ต้องการ

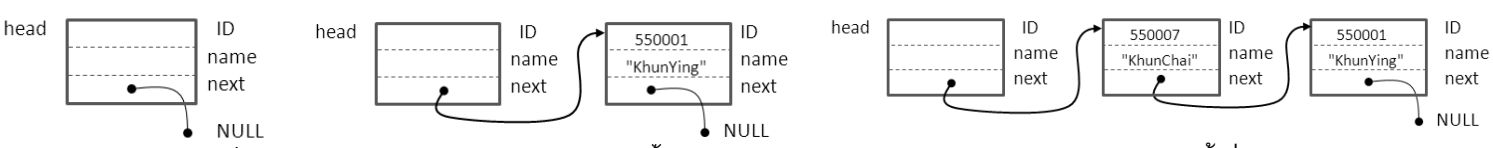
4. **โจทย์ตัวอย่าง:** เราจะเรียนรู้โครงสร้างข้อมูลแบบ linked list โดยการเขียนโปรแกรมตัวอย่างที่เกี่ยวกับระบบฐานข้อมูลของนิสิต โดยที่ operation พื้นฐานที่จำเป็น 3 อย่าง ได้แก่ การเพิ่ม (Add) การลบ (Delete) และการค้นหา (Find) ข้อมูลของนิสิตคนหนึ่งๆ ในฐานข้อมูล และกำหนดให้ข้อมูลของนิสิตคนหนึ่งประกอบด้วย รหัส และ ชื่อ (เพื่อความกระชับของโปรแกรมจึงขอแสดงตัวอย่างเพียง 2 ฟิลด์ย่อย แต่สามารถเพิ่มข้อมูลอื่นๆ เช่น วันเกิด, เพศ, เกรดเฉลี่ย เป็นต้น)

(4.1) จงทำโปรแกรมข้างล่างให้สมบูรณ์ โดยการเขียนโค้ดลงในฟังก์ชัน AddStudent ซึ่งจะทำการเพิ่ม node ใหม่ใน linked list โดยมีข้อมูลเท่ากับอาร์กิวเมนต์ของฟังก์ชัน โดย node ที่เพิ่มจะกลายเป็น node แรก

<u>Note</u>: เพื่อความสะดวกในการเขียนโปรแกรม เราจะกำหนดให้มี dummy head node (ตัวแปร dummy_head ในโปรแกรมข้างล่าง) ซึ่งเป็น node ที่ไม่เก็บข้อมูลอะไร ยกเว้นฟิลด์ next จะชี้ไปที่ node แรกของ linked list ใน Textbook หลายๆ เล่มจะไม่ได้ใช้ dummy head node แต่จะใช้ตัวแปรแบบ pointer เก็บแอดเดรสของ node แรกเลย

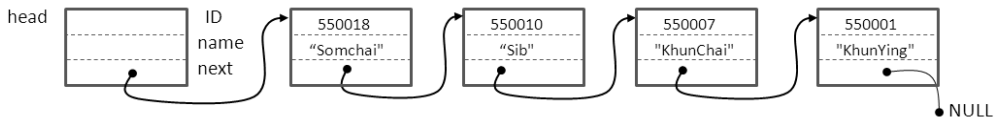
```
include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct s_node_tag S_NODE;
struct  s_node_tag {
  int ID;
  char name[21];
  S_NODE *next;
};

int AddStudent(S_NODE *head,int ID,char *name);

void main()
{
S_NODE dummy_head;
int retval;
   dummy_head.next=NULL;
   retval=AddStudent(&dummy_head,550001,"KhunYing");
   retval=AddStudent(&dummy_head,550007,"KhunChai");
   retval=AddStudent(&dummy_head,550010,"Sib");
   retval=AddStudent(&dummy_head,550018,"Somchai");

}
```

```
int AddStudent(S_NODE *head,int id,char *name)
{
S_NODE *new_s;

     // จองเนื้อที่สำหรับ node ใหม่

     ..................................................................

     if(new_s==NULL)
        return -1;

     // กำหนดข้อมูลในฟิลด์ย่อย ID และ name

     ..................................................................

     ..................................................................
// กำหนดข้อมูลในฟิลด์ย่อย next เพื่อใส่ node นี้ไว้เป็น node แรกของ Linked list

     ..................................................................

     ..................................................................

     return 0;
}
```



Linked list เริ่มต้น หลัง AddStudent ครั้งแรก หลัง AddStudent ครั้งที่ 2

(4.2) จงทำโปรแกรมข้างล่างให้สมบูรณ์ โดยการเขียนโค้ดลงในฟังก์ชัน FindStudent ซึ่งจะทำค้นหา node ที่เก็บข้อมูลของนิสิตที่มีรหัสตรงกับค่าอาร์กิวเมนต์ q_id ถ้าเจอ node ดังกล่าวให้ฟังก์ชันคืนค่า address หรือ pointer ที่ไปยัง node นั้น แต่ถ้าไม่เจอให้คืนค่า NULL

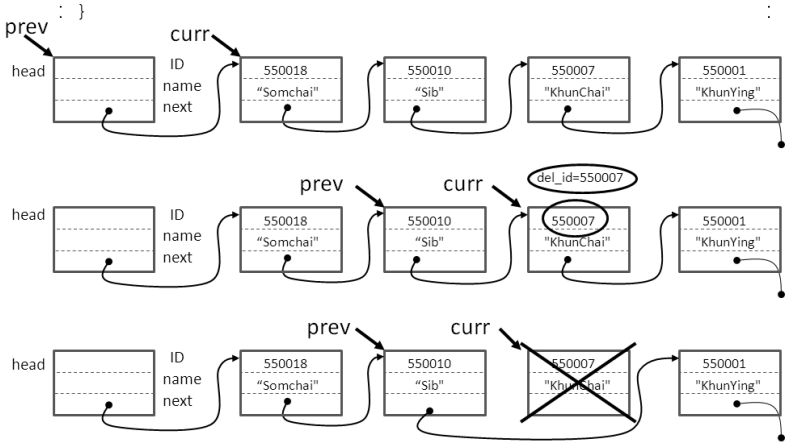



```
void main()
{
S_NODE dummy_head;
S_NODE *ptr;
int retval,query_id;
    ...
    ...
    retval=AddStudent(&dummy_head,550018,"Somchai");

     // ต่อจากโปรแกรมในหน้าที่แล้ว
    query_id=550010;
    ptr=FindStudent(&dummy_head,query_id);
    if(ptr!=NULL)
       printf("%d %s\n",ptr->ID,ptr->name);
    else
       printf("unknown student\n");
}
```

```
S_NODE *FindStudent(S_NODE *head,int q_id)
{
S_NODE *ptr;

      // กำหนดให้ ptr ชี้ไปที่ node แรกใน linked list
      ..............................................................................
      //  วนกำหนด ptr ให้ชี้ไปแต่ละโหนดใน linked list จนกว่าจะเจอ tail
      while(.............................................)
      {
        // ตรวจว่าข้อมูลในโหนดที่ชี้โดย ptr มีค่าเท่ากับ q_id ถ้าใช่คือ เจอ
          if(.....................................................)
                ...........................................................
        // เปลี่ยนค่า ptr ให้ชี้ไปที่โหนดถัดไป
            ..............................................................................
      ,}
      ..............................................................................
}
```

(4.2) จงทำโปรแกรมข้างล่างให้สมบูรณ์ โดยการเขียนโค้ดลงในฟังก์ชัน DeleteStudent ซึ่งจะทำลบ node ที่เก็บข้อมูลของนิสิตที่มีรหัสตรงกับค่าอาร์กิวเมนต์ del_id ในกรณีที่ไม่เจอ node ดังกล่าวให้ฟังก์ชันคืนค่า -1 มิฉะนั้นคืนค่า 0

```
void main()
{
S_NODE dummy_head;
S_NODE *ptr;
int retval,delete_id;
    ...
    ...
     // ต่อจากโปรแกรมในหน้าที่แล้ว
    delele_id=550007;
    retval=DeleteStudent(&dummy_head,delete_id);
}
```




```
S_NODE *DeleteStudent(S_NODE *head,int del_id)
{
S_NODE *prev,*curr;
      // กำหนดให้ curr ชี้ไปที่ node แรก และ prev ชี้ไปที่ node ก่อนหน้า curr
      ..............................................................................
      ..............................................................................
      //  วนกำหนด curr ให้ชี้ไปแต่ละโหนดใน linked list จนกว่าจะเจอ tail
      while(.............................................)
      { // ตรวจว่าข้อมูลในโหนดที่ชี้โดย curr มีค่าเท่ากับ del_id ถ้าใช่คือ เจอ
          if(.....................................................)
          {   // ปรับให้ prev ชี้ไปที่ node ถัดจาก curr
                ...........................................................
            // คืนหน่วยความจำที่จองไว้สำหรับโหนด curr
                ...........................................................
          }
          // เปลี่ยนค่า prev และ curr ให้ชี้ไปที่โหนดถัดไป
            ..............................................................................
            ..............................................................................
      ,}
     ..............................................................................
}
```

5. สรุปสิ่งที่ได้เรียนรู้เกี่ยวโครงสร้างข้อมูลแบบ linked list จากตัวอย่างโปรแกรมในหน้าที่แล้ว

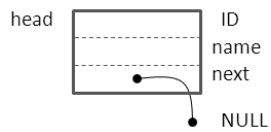
(i) การนิยาม structure จะมีฟิลด์ย่อยสำหรับเป็น pointer to โหนดถัดไป (โครงสร้างแบบ self-referential structures)

```
Syntax

typedef struct tag_name TYPE_NAME;
struct tag_name
{
      <type> <field_name_1>;
      <type> < field_name_2>;
      …
      TYPE_NAME *<next_field>;
}
```

```
Example

typedef struct s_node_tag S_NODE;
struct  s_node_tag
{
  int ID;
  char name[21];
  S_NODE *next;
};
```

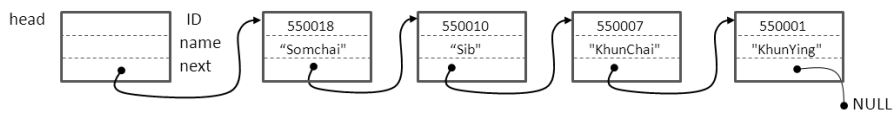
(ii) เราจะใช้ dummy head node ซึ่งไม่ได้เก็บข้อมูลอะไร แต่จะเอาไว้ชี้ไปที่ node แรกใน linked list โดยเริ่มต้นให้เท่ากับ NULL ซึ่งหมายถึง Linked list ไม่มีข้อมูลอะไร (ไม่ได้ชี้ไปที่ node ใด)

```
S_NODE head;
            head.next=NULL;
```



(iii) Operation พื้นฐานที่จำเป็น ได้แก่

- การเพิ่ม node ใหม่ -> AddStudent(..)
- การลบ node ที่มีอยู่แล้ว -> DeleteStudent(..)
- การค้นหา node -> FindStudent(..)



6. โจทย์ตัวอย่าง (ข้อสอบเก่าปี 2552): จงเขียนโค้ดในฟังก์ชัน `Add_Node` ซึ่งจะทำหน้าที่เพิ่ม Node ใหม่ที่มีข้อมูลเท่ากับค่าที่กำหนดโดยอาร์กิวเมนต์ `data` ลงใน Linked List ที่กำหนดโดยอาร์กิวเมนต์ `head_ptr`

โดยกำหนดให้การแทรก (insert) node ใหม่นี้ ต้องแทรกลงในตำแหน่ง Node แรกสุดของ Linked list เสมอ นอกจากนี้กำหนดให้การเพิ่ม Node ใหม่ จะทำได้ก็ต่อเมื่อข้อมูลที่กำหนดโดยอาร์กิวเมนต์ `data` ต้องไม่ซ้ำกับข้อมูลใน Node ใดๆ ที่มีอยู่แล้วของ Linked list (No duplicated data) ถ้าข้อมูลที่กำหนดโดยอาร์กิวเมนต์ `data` ซ้ำกับข้อมูลใน Node ใดๆ ที่มีอยู่แล้ว จะไม่มีการเปลี่ยนแปลงใดๆ ใน Linked list

```
typedef struct l_node LIST_NODE;
struct l_node {
       int data;
       LIST_NODE *next;
};
void Add_Node(LIST_NODE *head,int data)
{
LIST_NODE *new_node;
LIST_NODE *ptr;

     ptr=.........................................

     while(.........................................)
     {
         if(.........................................)
             return;
         .........................................
     } -- มีต่อ --
```

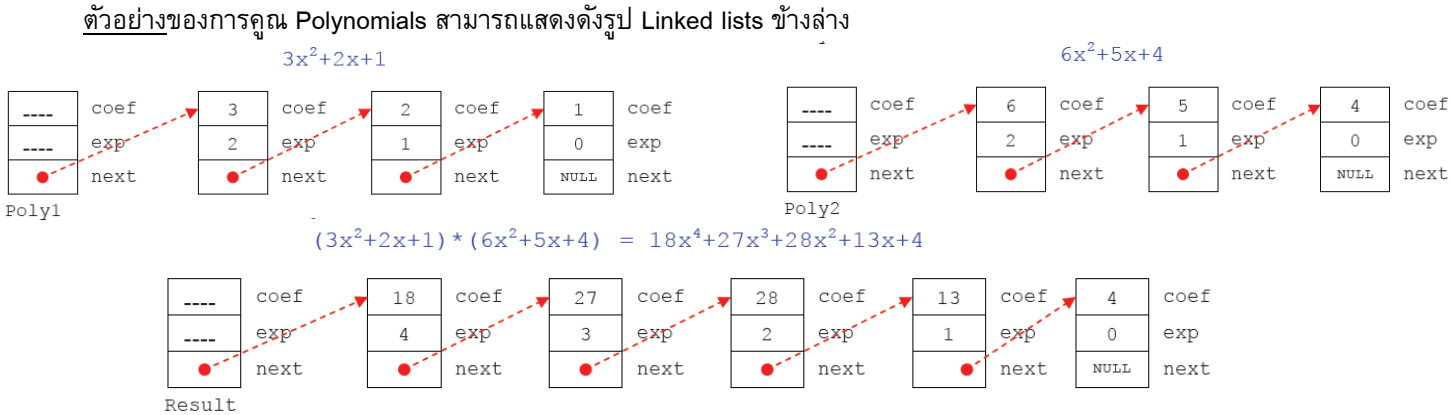
```
    new_node=.........................................

    if(.........................................)
        return;

    new_node->data=.........................................

    .........................................................

    .........................................................

    return;
},
```

7. โจทย์ตัวอย่าง Max_occurrence_2_Linked_List: กำหนดให้ข้อมูลนำเข้าเป็นจำนวนเต็มที่มีค่ามากกว่าหรือเท่ากับ 0 เข้ามาบรรทัดละตัวและสิ้นสุดด้วยจำนวนเต็มลบ จงเขียนโปรแกรมเพื่อนับว่าค่าใด ของข้อมูลนำเข้าถูกป้อนเข้ามามากที่สุด โดยให้แสดงค่าตัวเลข และจำนวนครั้งที่เจอออกสู่หน้าจอที่ละบรรทัด (ข้อนี้ขอบเขตของข้อมูลนำเข้าคือ 0-(2^{31}-1)) และจำนวนข้อมูลนำเข้าอาจจะมี 1 ตัว ถึง 1,000,000 ตัว <u>กำหนดให้การเขียนโปรแกรม</u>ใช้โครงสร้างข้อมูลแบบ linked list โดยที่แต่ละ Node นิยามโดย self-referenced structure `DATA_NODE` ดังโค้ดข้างล่าง

```
#include <stdio.h>
#include <stdlib.h>
typedef struct data_node_tag DATA_NODE;
struct  data_node_tag {
  int data;
  int count;
  DATA_NODE *next;
};
int Update_Count(DATA_NODE *head, int data);
int main()
{
DATA_NODE dummy_head,*ptr,*del_node;
int retval,max_count,max_data,x;
    dummy_head.next=NULL;
    while(1)
    {
        scanf("%d",&x);
        if(x<0)
            break;

// เพิ่มค่า count ของ x (ถ้า x ไม่มีอยู่ใน Linked list เพิ่ม node ใหม่แล้ว)

        Update_Count(&dummy_head,x);

    } – มีต่อ --
```

```
// วนลูปหา node ที่ count มากสุด
// และ free หน่วยความจำของแต่ละ node
    ptr=dummy_head.next;
    max_count=0;
    max_data=-1;
    while(ptr!=NULL)
    {
        if(ptr->count > max_count)
        {
            max_count=ptr->count;
            max_data=ptr->data;
        }
        del_node=ptr;
        ptr=ptr->next;
        free(del_node);
    }
    printf("%d\n%d\n",max_data,max_count);
    return 0;
}
int Update_Count(DATA_NODE *head, int data)
{
// วนลูปหา node ที่ฟิลด์ย่อย data ตรงกับ data แล้วเพิ่ม count
// ของ node นั้น ถ้าไม่เจอ node ดังกล่าว เพิ่ม node ใหม่
}
```

8. โจทย์ตัวอย่าง: จงเขียนฟังก์ชันชื่อ `Multiply_Polynomials` ซึ่งมีโครงของ Function Definition ดังข้างล่าง ฟังก์ชันนี้จะทำการคูณ polynomial ที่เก็บอยู่ในรูปของ Linked list ที่ระบุโดยอาร์กิวเมนต์ `Poly1_Head` กับ polynomial ที่ระบุโดยอาร์กิวเมนต์ `Poly2_Head` แล้วเก็บผลลัพธ์ลงใน polynomial ที่ระบุโดยอาร์กิวเมนต์ `Result_Head` โดยที่แต่ละ Node ใน linked list จะเป็นตัวแปรชนิด self-referenced structure `LIST_NODE` ซึ่งสามารถนิยามดังนี้

```
typedef struct l_node LIST_NODE;
struct l_node {
        double coef;            //  สัมประสิทธิ์
        int exp;                // เลขชี้กำลัง
        LIST_NODE *next;
};
int Multiply_Polynomials(LIST_NODE *Poly1_Head,LIST_NODE *Poly2_Head,LIST_NODE *Result_Head)
```

<u>ตัวอย่าง</u>ของการคูณ Polynomials สามารถแสดงดังรูป Linked lists ข้างล่าง