Recursion
การเรียกซ้ำ

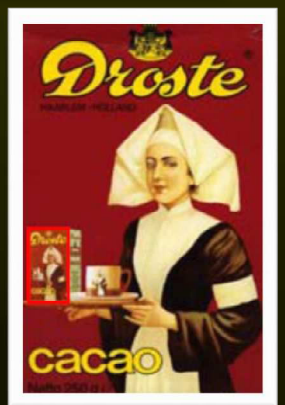
What is recursion?

Defining something *in terms of itself*

A "queue of people" is:

EMPTY
or
a person standing in
front of a "queue of people"

"Droste" effect



Recursion

- The process of solving a large problem by reducing it to one or more subprograms that are
  1. Identical in structure to the original problem
  2. Somewhat simpler to solve
- Use decompositional technique to divide each of these subprograms into new ones that are even less complex

ตัวอย่าง

```
void collect1000() {
    for (int i = 0; i<1000; i++) {
        Collect one baht from person i.
    }
}
```

```
void collect1000(int d) {
  if ()
    Contribute one baht directly.
  else {
    Find 10 people.
    Have each person collect d/10 baht.
    Return the money to your supervisor.
  }
}
```


การเรียกซ้ำ (Recursion)

- วิธีการที่ฟังก์ชันใดๆ สามารถเรียกตัวเองได้
- แต่ละครั้งที่ฟังก์ชันถูกเรียก จะเกิดตัวแปรอัตโนมัติชุดใหม่ที่ไม่เกี่ยวกับชุดเดิม
  - จึงเรียกวิธีนี้อีกชื่อหนึ่งว่า การเวียนเกิด
  - จึงใช้เนื้อที่ในหน่วยความจำมาก และทำงานช้า
- วิธีการเรียกซ้ำแบบนี้ ทำให้มีรหัสคำสั่งขนาดกะทัดรัด เขียนและเข้าใจง่าย

ลักษณะของฟังก์ชันแบบเวียนเกิด

- เป็นฟังก์ชันที่ต้องมีพารามิเตอร์
- แต่ละครั้งที่เรียกใช้ฟังก์ชันนั้น อาร์กิวเมนต์ของฟังก์ชัน(ข้อมูลที่ส่งให้กับฟังก์ชัน) จะง่ายขึ้น หรือซับซ้อนน้อยลง
- ฟังก์ชันแบบเวียนเกิดจะต้องมีกรณีจำกัดอย่างน้อย 1 กรณี
- เมื่ออาร์กิวเมนต์ของฟังก์ชันมีรูปแบบที่ง่ายที่สุด จะไม่มีความจำเป็นต้องเรียกตัวเอง เรียกกรณีนี้ว่า *กรณีจำกัด* ซึ่งเป็นกรณีที่ฟังก์ชันสามารถให้คำตอบได้โดยไม่ต้องเรียกตัวเองอีกนั่นเอง

แนวคิดในการออกแบบฟังก์ชันแบบเวียนเกิด

factorial(n)

```
5! = 5 * 4 * 3 * 2 * 1      = 5 * 4!
4! =     4 * 3 * 2 * 1      =     4 * 3!
3! =         3 * 2 * 1      =         3 * 2!
2! =             2 * 1      =             2 * 1!
1! =                 1
0! = 1
```

n! = n(n-1)!

Factorial(n) = n * Factorial(n-1)

อัลกอริทึม

MODULE factorial(N)
{ คำนวณค่าแฟคทอเรียลของ N}
IF N น้อยกว่าหรือเท่ากับ 1
THEN คำตอบคือ 1
ELSE คูณ N ด้วย factorial (N-1)
END MODULE

IF 3 <= 1 THEN .....
ELSE คูณ 3 ด้วย factorial (2) 6
IF 2 <= 1 THEN ...
ELSE คูณ 2 ด้วย factorial (1) 2
IF 1 <= 1 THEN คำตอบคือ 1 1
ELSE ...

factorial(n)

```
long int  factorial (int  n) {
   if (n <= 1)
      return  1;
   else
      return  (n * factorial (n - 1));
}
```

```
long int  factorial (int  n) {
   return (n <= 1? 1 : (n * factorial (n - 1));
}
```


ตัวอย่าง

```
/* Recursion not used */
#include <stdio.h>
void main () {
  int  n;

  for (n=10; n; n--)
     printf("%d ! ", n);
  printf("\n BLAST OFF\n");
}
```

10 ! 9 ! 8 ! 7 ! 6 ! 5 ! 4 ! 3 ! 2 ! 1 !
BLAST OFF

```
#include <stdio.h>
void count_down (int n) {
     if (n) {
       printf("%d ! ", n);
       /* The recursive call */
       count_down(n-1);
     }
     else
        printf("\nBLAST OFF\n");
}
void main() {
   /* The initial call */
    count_down(10);
}
```


ตัวอย่าง หาผลบวกแบบเรียกซ้ำ

```
/* Compute sums recursively*/
int sum (int n) {
  if (n <= 1)
     return  n;
  else
     return  n + sum(n-1);
}
```

Function call	*Value returned*
sum(1)	1
sum(2)	2 + sum(1) หรือ 2 + 1
sum(3)	3 + sum(2) หรือ 3 + 2 + 1
sum(4)	4 + sum(3) หรือ 4 + 3 + 2 + 1

Sum() function

```
int Sum(int n)
{
      if (n == 0)
            return 0;
      else
            return n + Sum(n-1);
}
```

base case

Sum() function

```
int main(void)
{
      int a = Sum(3);
      printf("Sum = %d\n", a);

      return 0;
}

int Sum(int n)
{
      if (n == 0)
            return 0;
      else
            return n + Sum(n-1);
}
```

main() a

Sum(3) n 3

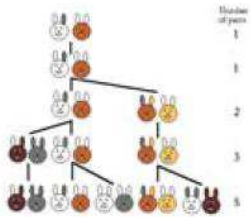
Sum(2) n 2

Sum(1) n 1

Sum(0) n 0

Fibonacci

1 1 2 3 5 8 13 21 ...



the 8th Fibonacci number

the 7th Fibonacci number + the 6th Fibonacci number

ตัวอย่าง ฟังก์ชันแบบเรียกซ้ำที่ผิดพลาด

```
/* Forgetting the base case*/
long factorial (long n) {
  return n * factorial(n-1);
}
```

```
/* Ambiguous use of decrement operator */
long factorial (long n) {
  if (n <= 1)
     return  1;
  else
     return n * factorial (- -n);
}
```

```
/* Incomplete base case test */
long factorial (long n) {
  if (n == 1)
     return  1;
  else
     return n * factorial(n-1);
}

factorial (0);
```


Linear Linked Lists

การดำเนินการพื้นฐานกับโครงสร้างข้อมูลแบบลิงค์ลิสต์ประกอบด้วย

- การสร้างลิสต์(Creating a list)
- การนับจำนวนโหนดในลิสต์(Counting the elements)
- การค้นหาโหนดในลิสต์(Looking up an element)
- การเพิ่มหรือแทรกโหนดเข้าลิสต์ (Inserting an element)
- การลบโหนดออกจากลิสต์ (Deleting an element)

การสร้างลิสต์

List.h

```
typedef   char    DATA
struct  linked_list  {
                                    DATA  data;
                                    struct linked_list * next;
};
typedef struct linked_list  NODE;
typedef NODE *            LINK;
```

```
// List creation by recursion
#include “list.h”
LINK  string2list (const char s[ ]) {
  LINK  head;
  if  (s[0] == NULL)
          return NULL;
  else {
          head = (LINK) malloc(sizeof(NODE));
          head->data = s[0];
          head->next = string2list(s+1);
          return head;
  }
}
```


การนับจำนวนโหนดในลิสต์

```
// Count elements in a list recursively
#include “list.h”
int count_elem(LINK  head) {

  if  (head == NULL)
    return 0;
  else
    return (1 + count_elem(head->next);
}
```

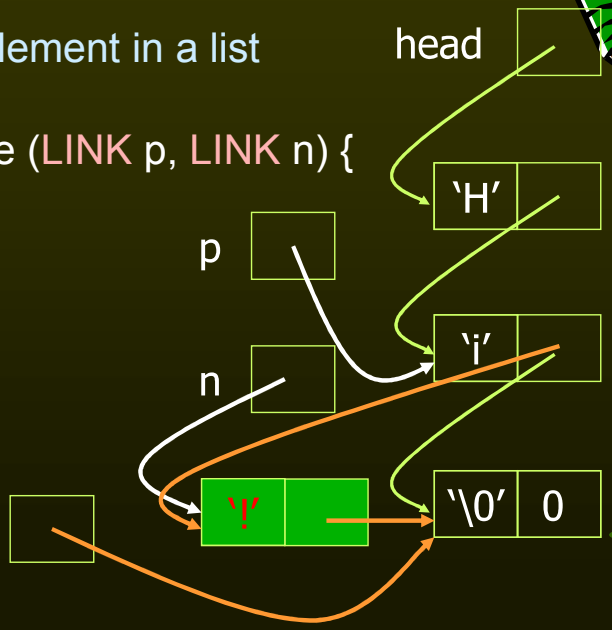

การค้นหาโหนดในลิสต์

```
// Looking up c element in the list pointed by
   head
#include “list.h”
LINK look_up(DATA c, LINK head) {
  if (head == NULL)
     return NULL;
  else  if (c == head->data)
            return head;
          else
            return (look_up(c, head->next));
}
```


การเพิ่มหรือแทรกโหนดเข้าลิสต์

```
// Inserting an element in a list
#include “list.h”
void insert_node (LINK p, LINK n) {
  LINK tmp;

tmp = p->next;
  p->next = n;
  n->next = tmp;
}
```



การลบโหนดออกจากลิสต์

```
// Deleting an element in the list
#include “list.h”
LINK  delete_node (LINK head, LINK d) {
    LINK nh=head, pd;
    if ((head != NULL) && (d != NULL)) {
        if (d == head) {
          nh = d->next;     free(d);  /* The first node is deleted */
       }
       else {
            while ((d != head) && (head != NULL)) {
                  pd = head;  head=head->next;
            }
            if (head) {
               pd->next = d->next;        free(d);
            }
       }
  }
  return nh;
}
```