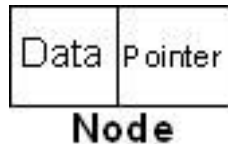
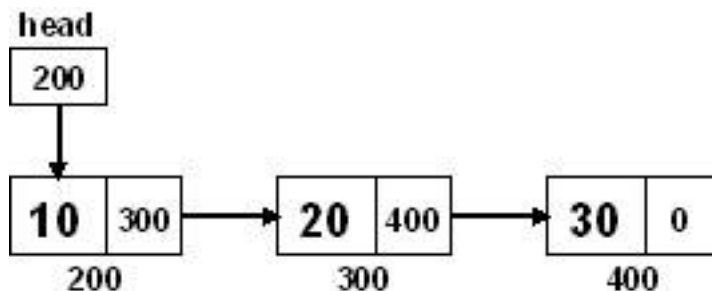


Linked List

- Linked List คือ โครงสร้างข้อมูลที่ประกอบด้วย Node ที่มาเชื่อมโยงกัน ซึ่งโครงสร้างแต่ละ Node ประกอบไปด้วยส่วนของ Data และ Pointer



- เมื่อ Node ต่างๆ มีการเชื่อมโยงกัน เราจำเป็นต้องมีตัวชี้เก็บค่า Address ของ (หรือชี้ไปที่) Node แรกเสมอ



- เราจะใช้ชนิดข้อมูลโครงสร้าง **struct** เพื่อกำหนดโครงสร้างของ Node รวมทั้งการจัดสรรหน่วยความจำแบบ Dynamic โดยใช้ **malloc** เพื่อจองพื้นที่หน่วยความจำให้แต่ละ Node ก่อนการใช้งาน แล้วจึงทำการเชื่อมโยง Node ต่างๆ เข้าด้วยกัน เมื่อไม่มีความต้องการใช้ Node แล้ว เราสามารถคืนพื้นที่หน่วยความจำด้วยการใช้ **free**
- เช่น หากต้องการเก็บข้อมูลนักศึกษาในกลุ่มหนึ่ง โดยใช้ **id** เก็บรหัส และ **name** เก็บชื่อ ซึ่งเราจะจัดเก็บเป็น Node เราสามารถนิยามโครงสร้าง Node ได้ ดังนี้

```
struct STUDENT {
    int id; char name[15]; struct STUDENT *next;
}
```

หมายความว่า ชนิดข้อมูลโครงสร้าง Node ประกอบด้วยฟิลด์ **id** เพื่อแทนรหัส ฟิลด์ **name** เพื่อแทนชื่อ และฟิลด์ตัวชี้ **next** เพื่อเก็บค่า Address ของ (หรือชี้ไปที่) Node ต่อไป

```
struct STUDENT st;
```

เป็นการประกาศตัวแปร **st** ที่มีชนิดข้อมูลโครงสร้าง **struct STUDENT**

หากเราต้องการสร้างนามแฝง หรือชื่อเล่นแทน **struct STUDENT** สามารถทำได้โดยใช้ **typedef** เช่น

```
typedef struct STUDENT SNODE;
```

ดังนั้น หากมีการประกาศตัวแปร **node1, node2** ให้มีชนิดข้อมูลโครงสร้าง **struct STUDENT** เราสามารถประกาศสองตัวแปรดังกล่าวได้ คือ

```
SNODE node1, node2;
```

จงพิจารณาการประกาศตัวชี้ **head**

```
SNODE *head;
```

เป็นการประกาศตัวชี้ **head** เพื่อบอกว่า ค่าข้อมูล ณ ตำแหน่ง Address ที่ **head** ชี้ไปมีชนิดข้อมูลโครงสร้าง **SNODE**

หากต้องการประกาศตัวชี้ชนิดข้อมูลโครงสร้างให้ง่ายขึ้น เราสามารถใช้ **typedef** มาช่วยได้ เช่น

```
typedef SNODE *SPTR;
```

เป็นการบอกว่า **SPTR** เป็นอีกชื่อหนึ่งของ **SNODE *** นั่นคือ

```
SPTR ptr;
```

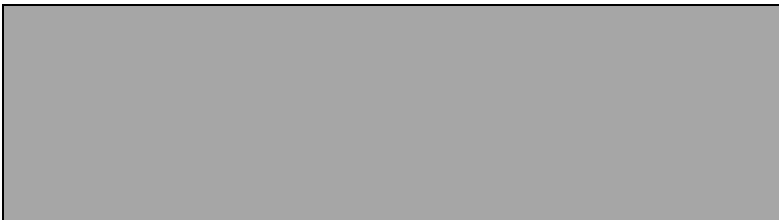
มีความหมายเดียวกับ

```
SNODE *ptr;
```

ตัวอย่างที่ 1

```
01 #include <iostream>
02 using namespace std;
03 struct NODE {
04     int code; struct NODE *next;
05 };
06
07 typedef struct NODE NODE;
08
09 int main()
10 {
11     NODE node1, node2, node3; NODE *ptr;
12     node1.code = 10; node1.next = &node2;
13     node2.code = 20; node2.next = &node3;
14     node3.code = 30; node3.next = NULL;
15
16     ptr = &node1;
17     while ( ptr != NULL ) {
18         cout<<ptr->code<<"->";
19         ptr = ptr->next;
20     }
21     cout<< endl;
22 }
```

Output

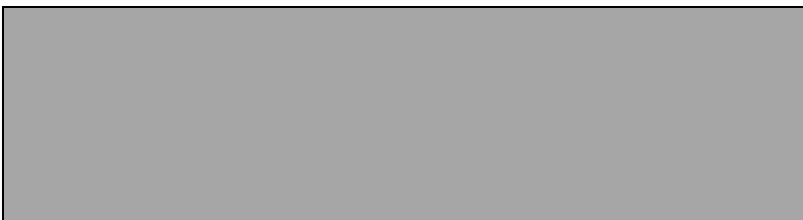


จาก ตัวอย่างที่ 1 จะเห็นว่า **node1**, **node2**, **node3** มีการจองพื้นที่หน่วยความจำให้กับตัวแปรแบบ Static หมายความว่า มีการจองพื้นที่หน่วยความจำก่อนการประมวลผล ซึ่งในการใช้งาน Linked List เราจะใช้การจองพื้นที่หน่วยความจำแบบ Dynamic

ตัวอย่างที่ 2

```
01 #include <iostream>
02 using namespace std;
03 struct NODE {
04     int code;
05     struct NODE *next;
06 };
07 typedef struct NODE NODE;
08 typedef struct NODE *PTR;
09
10 int main()
11 {
12     PTR head, newp, curr, prev, remv;
13
14     newp = new NODE ;
15     newp->code = 10; newp->next = NULL;
16     head = newp; prev = newp;
17     newp = new NODE;
18     newp->code = 20; newp->next = NULL;
19     prev->next = newp; prev = newp;
20
21     newp = new NODE;
22     newp->code = 30; newp->next = NULL;
23     prev->next = newp;
24
25     curr = head;
26     while ( curr != NULL ) {
27         cout<<"->"<<curr->code;
28         remv = curr; curr = curr->next;
29         delete( remv );
30     }
31     cout<<endl;
32 }
```

Output



จาก ตัวอย่างที่ 29 มีการจัดสรรพื้นที่หน่วยความจำแบบ Dynamic ให้กับตัวแปร รวมทั้งมีการคืนพื้นที่หน่วยความจำเมื่อไม่มีการใช้งานแล้ว

การดำเนินการกับ Linked List โดยทั่วไปจะเป็นการเพิ่มโหนดเข้าไปใน Linked List การค้นหา Node ที่ต้องการใน Linked List การลบ Node ใน Linked List ซึ่งอาจเป็นการลบบาง Node หรือลบ Node ทั้งหมดใน Linked และแสดงข้อมูลที่มีอยู่ใน Linked List

ตัวอย่างที่ 3

```
01 #include <iostream>
02 using namespace std;
03
04 struct NODE { int data; struct NODE *next; };
05
06 typedef struct NODE NODE; typedef NODE *PTR;
07
08 void insert( PTR *head_ref, int value ); void print_list( PTR ptr );
09
10 int main()
11 {
12     PTR head = NULL; int choice; int num;
13
14     cout<<"Press 1 to insert a number, Otherwise to Exit"<<endl;
15     cout<<"? " ; cin>> choice;
16     while ( choice == 1 ) {
17         cout<<"Enter a number : " ; cin>> num;
18         insert( &head, num ); print_list( head );
19         cout<<"Press 1 to insert a number, Otherwise to Exit"<<endl;
20         cout<<"? " ; cin>>choice;
21     }
22     cout<<"\n* * * End of Program * * *" <<endl;
23 }
24
25 void insert( PTR *head_ref, int value )
26 {
27     PTR newp, curr;
28     newp = new NODE ;
29     newp->data = value; newp->next = NULL; curr = *head_ref;
30     if ( curr == NULL )
31         *head_ref = newp;
32     else {
33         while ( curr->next != NULL )
34             curr = curr->next;
35         curr->next = newp;
36     }
37 }
38
39 void print_list( PTR ptr )
40 {
41     if ( ptr == NULL )
42         cout<<"List is empty"<<endl;
43     else {
44         cout<<"List is ";
45         while ( ptr != NULL ) {
46             cout<<"-> " << ptr->data;
47             ptr = ptr->next;
48         }
49         cout<<"NULL"<<endl;
50     }
51 }
```

Output – 1, 5, 1, 8, 1, 14, 1, 22, 0

```
Press 1 to insert a number, Otherwise to Exit
? 1
Enter a number : 5
List is 5 -> NULL

Press 1 to insert a number, Otherwise to Exit
? 1
Enter a number : 8
List is 5 -> 8 -> NULL

Press 1 to insert a number, Otherwise to Exit
? 1
Enter a number : 14
List is 5 -> 8 -> 14 -> NULL

Press 1 to insert a number, Otherwise to Exit
? 1
Enter a number : 22
List is 5 -> 8 -> 14 -> 22 -> NULL

Press 1 to insert a number, Otherwise to Exit
? 0

* * * End of Program * * *
```

หลังจากทราบแล้วว่า Linked List คือ อะไร และมีการเก็บข้อมูลอย่างไร ต่อไปจะมาเรียนรู้เกี่ยวกับการดำเนินการกับ Linked List ซึ่งโดยทั่วไปจะมีการดำเนินการ ดังนี้

- Linked List ว่างหรือไม่ (empty)
- การเพิ่ม Node ต่อท้าย Linked List (append)
- การแทรก Node ณ ตำแหน่งที่ต้องการ (insert)
- การหาขนาดของ Linked List (length)
- การท่องไปยัง Node ต่างๆ ใน Linked List เพื่อแสดงผลออกมา (print_list)
- การลบ Node บาง Node ใน Linked List (delete node)
- การลบ Node ทั้งหมดใน Linked List (delete list)

ตัวอย่างที่ 4

```
01 #include <iostream>
02 using namespace std;
03
04 struct STUDENT {
05     int id; char name[15]; float score; struct STUDENT *next;
06 };
07
08 typedef struct STUDENT SNODE; typedef SNODE *SPTR;
09
10 void display_menu();          SNODE get_new_st();
11 int empty( SPTR ptr );        int length( SPTR ptr );
12 void print_list( SPTR ptr );  void append( SPTR *head_ref );
13 void insert( SPTR *head_ref ); void del_node( SPTR *head_ref );
14 void del_list( SPTR *head_ref );
15
16 int main()
17 {
18     SPTR head = NULL; SNODE new_st; int choice;
19
20     display_menu();
21     cout<<endl<<"Choice: "; cin>>choice;
22
23     while ( choice ) {
24         switch( choice ) {
25             case 1: append( &head ); break;
26             case 2: insert( &head ); break;
27             case 3: del_node( &head ); break;
28             case 4: del_list( &head ); break;
29             case 5: print_list( head ); break;
30             default: cout<<"Invalid Choice!!!"<<endl; display_menu();
31         }
32         cout<<"Choice: "; cin>>choice;
33     }
34     cout<<endl<<"* * * End of Program * * *"<<endl;
35 }
36
37 void display_menu()
38 {
39     cout<<"Enter Your Choice"<<endl; cout<<"-----"<<endl;
40     cout<<"0. Exit"<<endl;          cout<<"1. Append Student"<<endl;
41     cout<<"2. Insert Student"<<endl; cout<<"3. Delete Student"<<endl;
42     cout<<"4. Delete All"<<endl;      cout<<"5. Print List"<<endl;
43 }
44
45 SNODE get_new_st()
46 {
47     SNODE st;
48
49     cout<<"ID Name Score: " ;
50     cin>>st.id>>st.name>>st.score;
51     st.next = NULL;
52
53     return st;
54 }
55
56 int empty( SPTR ptr )
57 {
58     return ptr == NULL;
59 }
```

```
60
61 int length( SPTR ptr )
62 {
63     int count = 0;
64
65     while ( !empty( ptr ) ) {
66         count++; ptr = ptr->next;
67     }
68
69     return count;
70 }
71
72 void print_list( SPTR ptr )
73 {
74     int size = length( ptr );
75
76     cout<<"List: ";
77     while ( !empty( ptr ) ) {
78         cout<<ptr->id<<" , "<<ptr->name<<" , "<<ptr->score<<"->" ;
79         ptr = ptr->next;
80     }
81     cout<<"NULL"<<endl; cout<< "Size of List: "<< size ;
82 }
83
84 void append( SPTR *head_ref )
85 {
86     SPTR curr = *head_ref, newp; SNODE s;
87
88     s = get_new_st(); newp = new SNODE;
89     cout<<"Append Student"<<endl;
90     newp->id = s.id;      strcpy( newp->name, s.name );
91     newp->score = s.score; newp->next = NULL;
92
93     if ( empty( curr ) )
94         *head_ref = newp;
95     else {
96         while ( !empty( curr->next ) )
97             curr = curr->next;
98         curr->next = newp;
99     }
100 }
101
102 void insert( SPTR *head_ref )
103 {
104     SPTR curr = *head_ref, newp; SNODE s; int index, i;
105
106     cout<<"Index : " ; cin>>index;
107     if ( index > length( curr ) - 1 )
108         cout<<"Index is Too Big!!!"<<endl;
109     else {
110         s = get_new_st(); newp = new SNODE ;
111         cout<<"Insert Student at Index"<< index ;
112         newp->id = s.id;      strcpy( newp->name, s.name );
113         newp->score = s.score; newp->next = NULL;
114         if ( index == 0 ) {
115             *head_ref = newp; newp->next = curr;
116         } else {
117             for ( i = 0; i < index - 1; i++ )
118                 curr = curr->next;
119             newp->next = curr->next; curr->next = newp;
120         }
121     }
```



```
122 }
123
124 void del_node( SPTR *head_ref )
125 {
126     SPTR prev, curr, ptr; int id;
127
128     cout<<"ID : "; cin>>id;
129
130     if ( id == ( *head_ref )->id ) {
131         ptr = *head_ref; *head_ref = ( *head_ref )->next;
132         delete( ptr );      cout<<id<<"is deleted"<<endl;
133     } else {
134         prev = *head_ref; curr = ( *head_ref )->next;
135         while ( !empty( curr ) && curr->id != id ) {
136             prev = curr; curr = curr->next;
137         }
138         if ( !empty( curr ) ) {
139             ptr = curr; prev->next = curr->next;
140             delete( ptr ); cout<<id<<" is deleted"<<endl;
141         } else
142             cout<<id<<" is not found!!!"<<endl;
143     }
144 }
145
146 void del_list( SPTR *head_ref )
147 {
148     SPTR curr = *head_ref, ptr;
149
150     while ( !empty( curr ) ) {
151         ptr = curr->next; delete( curr ); curr = ptr;
152     }
153     *head_ref = NULL;
154     cout<<"Delete All!!!"<<endl;
155 }
```

Output

```
Enter Your Choice
-----
0. Exit
1. Append Student
2. Insert Student
3. Delete Student
4. Delete All
5. Print List

Choice: 5
List: NULL
Size of List: 0

Choice: 1
ID Name Score: 101 Joe 57.8
Append Student

Choice: 5
List: [101,Joe,57.80] -> NULL
Size of List: 1

Choice: 1
ID Name Score: 102 Ann 72.8
Append Student
```

```
Choice: 5
List: [101,Joe,57.80] -> [102,Ann,72.80] -> NULL
Size of List: 2

Choice: 2
Index : 0
ID Name Score: 201 Tim 69.4
Insert Student at Index 0

Choice: 5
List: [201,Tim,69.40] -> [101,Joe,57.80] -> [102,Ann,72.80] -> NULL
Size of List: 3

Choice: 1
ID Name Score: 202 Doe 86.3
Append Student

Choice: 5
List: [201,Tim,69.40] -> [101,Joe,57.80] -> [102,Ann,72.80] -> [202,Doe,86.30]
-> NULL
Size of List: 4

Choice: 9
Invalid Choice!!!

Enter Your Choice
-----
0. Exit
1. Append Student
2. Insert Student
3. Delete Student
4. Delete All
5. Print List

Choice: 2
Index : 3
ID Name Score: 301 Bob 63.8
Insert Student at Index 3

Choice: 5
List: [201,Tim,69.40] -> [101,Joe,57.80] -> [102,Ann,72.80] -> [301,Bob,63.80]
-> [202,Doe,86.30] -> NULL
Size of List: 5

Choice: 1
ID Name Score: 302 Pat 92.0
Append Student

Choice: 5
List: [201,Tim,69.40] -> [101,Joe,57.80] -> [102,Ann,72.80] -> [301,Bob,63.80]
-> [202,Doe,86.30] -> [302,Pat,92.00] -> NULL
Size of List: 6

Choice: 2
Index : 6
Index is Too Big!!!

Choice: 5
List: [201,Tim,69.40] -> [101,Joe,57.80] -> [102,Ann,72.80] -> [301,Bob,63.80]
-> [202,Doe,86.30] -> [302,Pat,92.00] -> NULL
Size of List: 6
```

```
Choice: 1
ID Name Score: 104 Sam 76.9
Append Student

Choice: 5
List: [201,Tim,69.40] -> [101,Joe,57.80] -> [102,Ann,72.80] -> [301,Bob,63.80]
-> [202,Doe,86.30] -> [302,Pat,92.00] -> [104,Sam,76.90] -> NULL
Size of List: 7

Choice: 8
Invalid Choice!!!

Enter Your Choice
-----
0. Exit
1. Append Student
2. Insert Student
3. Delete Student
4. Delete All
5. Print List

Choice: 2
Index : 10
Index is Too Big!!!

Choice: 3
ID : 101
101 is deleted

Choice: 5
List: [201,Tim,69.40] -> [102,Ann,72.80] -> [301,Bob,63.80] -> [202,Doe,86.30]
-> [302,Pat,92.00] -> [104,Sam,76.90] -> NULL
Size of List: 6

Choice: 3
ID : 401
401 is not found!!!

Choice: 5
List: [201,Tim,69.40] -> [102,Ann,72.80] -> [301,Bob,63.80] -> [202,Doe,86.30]
-> [302,Pat,92.00] -> [104,Sam,76.90] -> NULL
Size of List: 6

Choice: 3
ID : 201
201 is deleted

Choice: 5
List: [102,Ann,72.80] -> [301,Bob,63.80] -> [202,Doe,86.30] -> [302,Pat,92.00]
-> [104,Sam,76.90] -> NULL
Size of List: 5

Choice: 7
Invalid Choice!!!

Enter Your Choice
-----
0. Exit
1. Append Student
2. Insert Student
3. Delete Student
4. Delete All
```

5. Print List

Choice: 3
ID : 104
104 is deleted

Choice: 5
List: [102,Ann,72.80] -> [301,Bob,63.80] -> [202,Doe,86.30] -> [302,Pat,92.00]
-> NULL
Size of List: 4

Choice: 3
ID : 301
301 is deleted

Choice: 5
List: [102,Ann,72.80] -> [202,Doe,86.30] -> [302,Pat,92.00] -> NULL
Size of List: 3

Choice: 4
Delete All!!!

Choice: 5
List: NULL
Size of List: 0

Choice: 2
Index : 0
Index is Too Big!!!

Choice: 1
ID Name Score: 401 Dan 84.7
Append Student

Choice: 5
List: [401,Dan,84.70] -> NULL
Size of List: 1

Choice: 6
Invalid Choice!!!

Enter Your Choice

0. Exit
1. Append Student
2. Insert Student
3. Delete Student
4. Delete All
5. Print List

Choice: 4
Delete All!!!

Choice: 5
List: NULL
Size of List: 0

Choice: 0

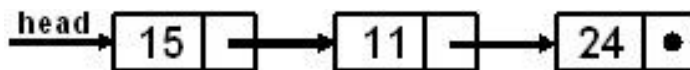
* * * End of Program * * *

ตามที่ทราบมาแล้วว่า Linked List คือ การเชื่อมโยงของ Node ต่างๆ เข้าด้วยกัน แล้วมีตัวชี้ไปที่ Node แรกเสมอ ซึ่ง Linked List สามารถแบ่งออกได้เป็น 2 ประเภทใหญ่ๆ คือ

1. Linear Linked List (Linked List แบบเส้นตรง) ซึ่งแบ่งย่อยได้อีก 2 ประเภท คือ

1.1 Single Linked List (Linked List แบบเดี่ยว)

เป็นแบบที่เราได้เรียนรูมาแล้วก่อนหน้านี้ นั่นคือ เป็น Linked List ที่มี 1 Link ต่อ Node (Link ดังกล่าว คือ Field ตัวชี้ **next** ของ Node นั้นเอง) Link เดี่ยวของแต่ละ Node จะชี้ไปยัง Node ต่อไป ใน List หรือ Link เป็น **NULL** (ไม่ชี้ไปที่ไหน) ในกรณีที่ เป็น Node สุดท้ายของ List



1.2 Double Linked List (Linked List แบบคู่)

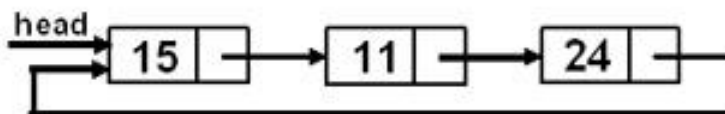
แบบนี้ แต่ละ Node จะมี 2 Link โดยที่ จะมี Link หนึ่งชี้ไปยัง Node ก่อนหน้านี้ (มักจะตั้งชื่อฟิลด์ตัวชี้ว่า **prev**) หรือ Link เป็น **NULL** ถ้าเป็น Node แรก ของ List ส่วนอีก Link (ชื่อว่า **next**) ชี้ไปยัง Node ต่อไป หรือ Link เป็น **NULL** ถ้าเป็น Node สุดท้าย ของ List



2. Circular Linked List (Linked List แบบวงกลม) เป็นแบบที่ Node แรก และ Node สุดท้าย Link เข้าหากัน ซึ่งทำได้ทั้งแบบ Single และ Double Linked List การท่องเที่ยวไปยังโหนดต่างๆ ใน Linked List สามารถเริ่มต้นที่ Node ใดก็ได้ ในทิศทางใดก็ได้จนกว่าจะท่องเที่ยวกลับมาถึง Node ต้นทาง ถ้ามองอีกมุมหนึ่ง จะพบว่า Circular Linked List จะไม่มี Node เริ่มต้น และ Node สิ้นสุด

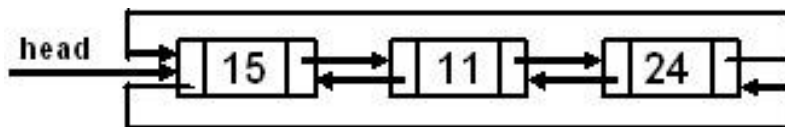
2.1 Circular Single Linked List (Linked List วงกลมแบบเดี่ยว)

คล้ายกับ Single Linked List ยกเว้นเพียงแต่ว่า Link ของโหนดสุดท้ายจะชี้ไปยัง Node แรก



2.2 Circular Double Linked List (Linked List วงกลมแบบคู่)

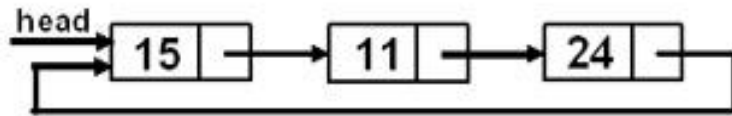
คล้ายกับ Double Linked List ยกเว้นเพียงแต่ว่า ที่ Node แรก Link ที่ชี้ไปที่ Node ก่อนหน้านี้ (**prev**) จะชี้ไปยัง Node สุดท้าย และที่ Node สุดท้าย Link ที่ชี้ไปยัง Node ถัดไป (**next**) ชี้ไปที่ Node แรก



ตัวอย่างที่ 5 กำหนดให้มี

```
struct NODE { int data; struct NODE *next; };  
typedef struct NODE NODE; typedef NODE * NODE_PTR;
```

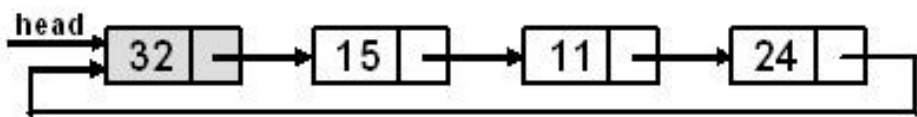
- A. ให้เขียนชุดคำสั่งเพื่อให้ได้ Linked List ดังรูปด้านล่าง โดยให้มีการประกาศตัวชี้ **head** เพียงตัวเดียว
(**NODE_PTR head;**)



- B. ให้มีการประกาศตัวชี้ **NODE** เพิ่มชื่อ **newp** และ **curr** (**NODE_PTR newp, curr;**)
จาก Linked List ในข้อ A ให้เขียนคำสั่งเพื่อสร้างโหนดใหม่ขึ้นมา ดังนี้

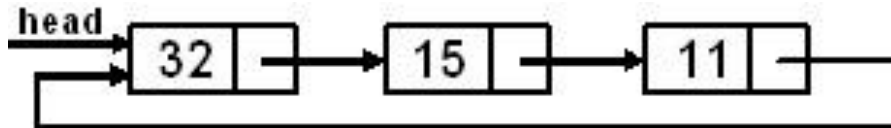


แล้วให้เขียนชุดคำสั่งเพื่อทำการนำ Node ใหม่ดังกล่าวแทรกที่ Node แรก แล้วจะได้ Linked List ดังรูปด้านล่าง



C. ให้มีการประกาศตัวชี้ **NODE** เพิ่มอีกชื่อ **prev** (**NODE_PTR prev;**)

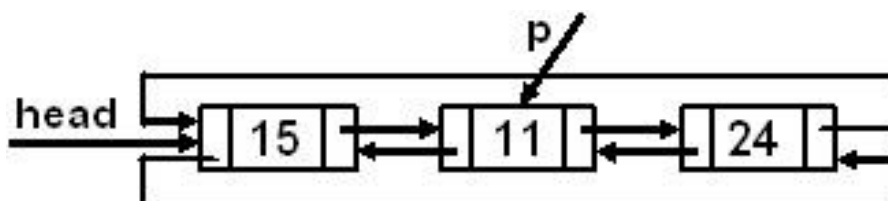
จากข้อ B ให้เขียนชุดคำสั่งเพื่อลบ Node สุดท้ายออก แล้วจะได้ Linked List ดังรูปด้านล่าง (ห้ามใช้ข้อมูลในฟิลด์ **data** ในการอ้างอิง เพราะในความเป็นจริง เราอาจไม่ทราบว่าโหนดสุดท้ายมีข้อมูลอะไรอยู่ เพียงแต่สิ่งที่เราต้องการ คือ ลบ Node สุดท้ายออก)



ตัวอย่างที่ 6 กำหนดให้มี

```
struct NODE { int data; struct NODE *prev, *next; };  
typedef struct NODE NODE; typedef NODE * NODE_PTR;
```

A. ให้เขียนชุดคำสั่งเพื่อให้ได้ Linked List ดังรูปด้านล่าง โดยมีการประกาศตัวชี้ **head**, **p** (**NODE_PTR head, p;**)

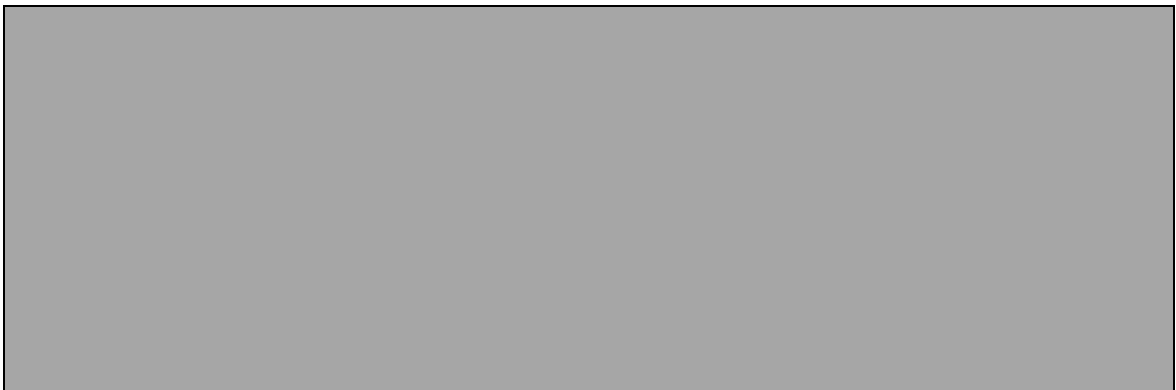
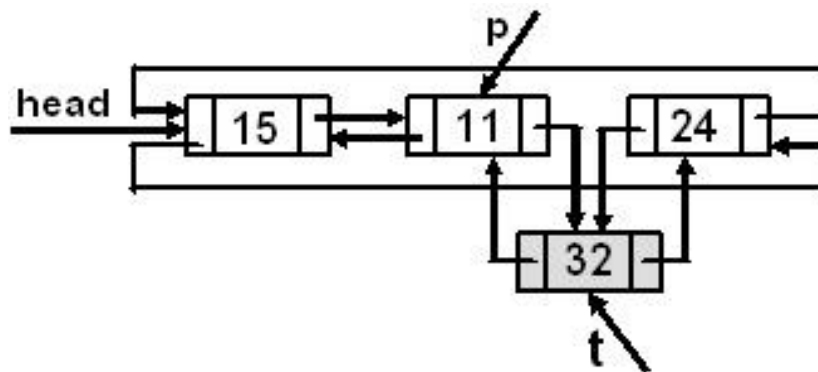




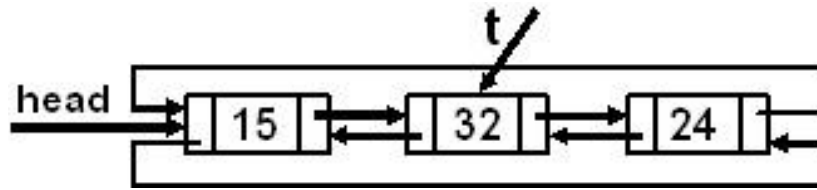
- B. ให้มีการประกาศตัวชี้ **NODE** เพิ่มชื่อ **t** (**NODE_PTR t;**) จาก Linked List ในข้อ A ให้เขียนคำสั่งเพื่อสร้าง Node ใหม่ขึ้นมา ดังนี้

32

แล้วให้เขียนชุดคำสั่งเพื่อทำการนำ Node ใหม่ แล้วให้ได้ Linked List ดังรูปด้านล่าง



C. จากข้อ B ให้เขียนชุดคำสั่งเพื่อลบ Node ที่ **p** ชี้ออก แล้วจะได้ Linked List ดังรูปด้านล่าง



Linked list STL

ไลบรารีแม่แบบมาตรฐาน (อังกฤษ: Standard Template Library / STL) เป็นไลบรารีของภาษาซีพลัสพลัส ประกอบไปด้วยคลาสของขั้นตอนวิธี คอนเทนเนอร์ (โครงสร้างข้อมูลและชนิดข้อมูล) ฟังก์เตอร์ และ ตัววนซ้ำ [1] ไลบรารีแม่แบบมาตรฐานของ ISO C++ ได้อ้างอิงตามไลบรารีแม่แบบมาตรฐานของ Silicon Graphics (SGI)[ต้องการอ้างอิง]

คอนเทนเนอร์[แก้]

คอนเทนเนอร์ คือ โครงสร้างข้อมูล และชนิดข้อมูล ประกอบไปด้วย **รายการลำดับ** ได้แก่ vector deque list **คอนเทนเนอร์แบบจับคู่** ได้แก่ set multiset map multimap คอนเทนเนอร์ดัดแปลงได้แก่ priority_queue queue stack และคอนเทนเนอร์ประเภทอื่นๆ

คอนเทนเนอร์	รายละเอียด
คอนเทนเนอร์อย่างง่าย	
pair	คู่อันดับ เป็นคอนเทนเนอร์อย่างง่าย เป็นคู่อันดับของวัตถุซึ่งเรียกว่า first และ second คู่อันดับสามารถกำหนดค่า คัดลอกค่า และเปรียบเทียบได้ คอนเทนเนอร์คู่อันดับนี้มีลักษณะคล้าย คู่อันดับ ในคณิตศาสตร์ สำหรับคอนเทนเนอร์ map และ hash_map (รายละเอียดอยู่ด้านล่าง) ข้อมูลแต่ละตัวจะเป็นคู่อันดับ โดยที่ first เป็นคีย์ ส่วน second เป็นข้อมูล
รายการลำดับ	
vector	เป็น แถวลำดับ ที่สามารถปรับขนาดได้ มีความสามารถในการ เข้าถึงโดยสุ่ม เหมือนอาร์เรย์ทั่วไป แต่สามารถปรับขนาดได้เมื่อมีการเพิ่มหรือลดข้อมูล โดยทั่วไปเวกเตอร์ จะทำการจองพื้นที่มากกว่าข้อมูลที่มีอยู่ เมื่อมีการเพิ่มข้อมูลจนเต็มเวกเตอร์ก็จะทำการจองพื้นที่เพิ่มเป็น 2 เท่าของพื้นที่เดิม การกระทำเช่นนี้เมื่อ ถ่วงเฉลี่ย ออกมาแล้วจะได้ว่าการเพิ่ม/ลดข้อมูลในแต่ละครั้งทางปลายด้านหลัง ใช้ เวลาคงตัว (ส่วนการเพิ่ม/ลบข้อมูล ณ ตำแหน่งใดๆใช้เวลาแปรผันตามจำนวนข้อมูล) สำหรับ เวกเตอร์ ของ ชนิดข้อมูลแบบบูล จะมีการปรับแต่งให้ใช้เนื้อที่เพียงแค่ 1

	บิตต่อสมาชิก 1 ตัวเท่านั้น
list	เป็นรายการโยง 2 ทาง สมาชิกแต่ละตัวไม่ได้มีพื้นที่ในหน่วยความจำติดกัน เหมือนกับอาร์เรย์ มีประสิทธิภาพตรงกันข้ามกับอาร์เรย์ กล่าวคือ การเข้าถึงข้อมูล ใช้เวลาแปรผันตามจำนวนข้อมูล แต่การเพิ่ม/ลบข้อมูล ณ ตำแหน่งใดๆ ใช้เวลาคงตัว
deque	เหมือนรายการโยง สามารถเข้าถึงข้อมูลและเพิ่ม / ลบข้อมูลที่ปลายทั้งด้านหน้า และด้านหลังได้ภายในเวลาคงตัว
คอนเทนเนอร์ที่ดัดแปลง	
queue	เป็นคอนเทนเนอร์ที่มีลักษณะแบบ เข้าก่อนออกก่อน สามารถเพิ่มข้อมูลทางด้านหลัง และนำข้อมูลออกทางด้านหน้า
priority_queue	เป็นแถวคอยซึ่งมีการเรียงโดยอัตโนมัติ ข้อมูลที่สำคัญที่สุดจะอยู่บนสุดและจะถูกนำเอาออกเป็นอันดับแรก
stack	เป็นคอนเทนเนอร์แบบ เข้าทีหลังออกก่อน สามารถเพิ่มข้อมูลทางด้านหลัง และนำข้อมูลออกทางด้านหลังด้วยเช่นกัน
คอนเทนเนอร์แบบจับคู่	
set	เป็นคอนเทนเนอร์ชนิดหนึ่งที่สมาชิกห้ามซ้ำกัน มีความสามารถเหมือน เซต กล่าวคือสามารถยูเนียน อินเตอร์เซกชัน หาผลต่าง หาผลต่างสมมาตร และทดสอบการเป็นสมาชิกได้ โดยปกติจะอิมพลีเมนต์เซตด้วยต้นไม้ค้นหาแบบ ทวิภาคที่มีโครงสร้างปรับสมดุลเองได้ ดังนั้นจึงจำเป็นที่ข้อมูลจะต้องสามารถถูกเปรียบเทียบได้ (ด้วย โอเปอเรเตอร์ < หรือการกำหนดฟังก์ชันเปรียบเทียบ) นอกจากนี้ข้อมูลยังต้องมีลักษณะ strict weak ordering ไม่เช่นนั้นอาจทำให้เกิดข้อผิดพลาดขึ้นได้
multiset	เหมือนเซต แต่สมาชิกมีข้อมูลซ้ำกันได้
map	เป็นแถวลำดับแบบจับคู่ ซึ่งสามารถจับคู่จากคีย์ไปยังข้อมูลได้ ดังนั้นคีย์จึงห้ามซ้ำกัน ข้อมูลจะถูกเก็บเป็นคู่อันดับโดยที่ first เป็นคีย์ ส่วน second เป็นข้อมูล เช่นเดียวกับเซต แมพอิมพลีเมนต์โดยใช้ต้นไม้ค้นหาแบบทวิภาคที่มีโครงสร้าง ปรับสมดุลเองได้ ดังนั้นจึงจำเป็นที่ข้อมูลจะต้องสามารถถูกเปรียบเทียบได้ (ด้วย ตัวดำเนินการ < หรือการกำหนดฟังก์ชันเปรียบเทียบ) นอกจากนี้ข้อมูลยังต้องมีลักษณะ strict weak ordering ไม่เช่นนั้นอาจทำให้เกิดข้อผิดพลาดขึ้นได้
multimap	เหมือนแมพ แต่มีคีย์ซ้ำกันได้
hash_set	คล้ายเซต มัลติเซต แมพ มัลติแมพ แต่การอิมพลีเมนต์กลับใช้ตารางแฮชแทน

hash_multiset hash_map hash_multimap	<p>ต้นไม้ค้นหาแบบทวิภาคที่มีโครงสร้างปรับสมดุลเองได้ ซึ่งในตารางแฮช ก็จะไม่เรียง และไม่จำเป็นต้องมีตัวดำเนินการเพื่อเปรียบเทียบข้อมูลด้วย แต่ต้องการฟังก์ชันแฮชแทน ข้อได้เปรียบของการใช้ตารางแฮชแทนคือสามารถเข้าถึงข้อมูลโดยคาดหวังให้ใช้เวลาคงที่ได้ คอนเทนเนอร์เหล่านี้ไม่ได้รวมอยู่ในมาตรฐานภาษาซีพลัสพลัส แต่อยู่ในแม่แบบมาตรฐานของ SGI สำหรับ C++11 ซึ่งจะเป็นมาตรฐานภาษาซีพลัสพลัสรุ่นใหม่มีการใช้ unordered_set, unordered_multiset, unordered_map และ unordered_multimap แทนแฮชเซต, แฮชมัลติเซต, แฮชแมพ และแฮชมัลติแมพ ตามลำดับ</p>
--	---

ประกาศ Header #include <list> และเลือกใช้ object ต่างๆดังนี้

TABLE 5-9 Various ways to declare a list object

Statement	Description
<code>list<elemType> listCont;</code>	Creates the empty list container listCont. (The default constructor is invoked.)
<code>list<elemType> listCont(otherList);</code>	Creates the list container listCont and initializes it to the elements of otherList. listCont and otherList are of the same type.
<code>list<elemType> listCont(size);</code>	Creates the list container listCont of size size. listCont is initialized using the default constructor.
<code>list<elemType> listCont(n, elem);</code>	Creates the list container listCont of size n. listCont is initialized using n copies of the element elem.
<code>list<elemType> listCont(beg, end);</code>	Creates the list container listCont. listCont is initialized to the elements in the range [beg, end), that is, all the elements in the range beg...end-1. Both beg and end are iterators.

TABLE 5-10 Operations specific to a list container

Expression	Description
<code>listCont.assign(n, elem)</code>	Assigns <code>n</code> copies of <code>elem</code> .
<code>listCont.assign(beg, end)</code>	Assigns all the elements in the range <code>beg...end-1</code> . Both <code>beg</code> and <code>end</code> are iterators.
<code>listCont.push_front(elem)</code>	Inserts <code>elem</code> at the beginning of <code>listCont</code> .
<code>listCont.pop_front()</code>	Removes the first element from <code>listCont</code> .
<code>listCont.front()</code>	Returns the first element. (Does not check whether the container is empty.)
<code>listCont.back()</code>	Returns the last element. (Does not check whether the container is empty.)
<code>listCont.remove(elem)</code>	Removes all the elements that are equal to <code>elem</code> .
<code>listCont.remove_if(oper)</code>	Removes all the elements for which <code>oper</code> is true.
<code>listCont.unique()</code>	If the consecutive elements in <code>listCont</code> have the same value, removes the duplicates.
<code>listCont.unique(oper)</code>	If the consecutive elements in <code>listCont</code> have the same value, removes the duplicates, for which <code>oper</code> is true.
<code>listCont1.splice(pos, listCont2)</code>	All the elements of <code>listCont2</code> are moved to <code>listCont1</code> before the position specified by the iterator <code>pos</code> . After this operation, <code>listCont2</code> is empty.

TABLE 5-10 Operations specific to a `list` container (continued)

Expression	Description
<code>listCont1.splice(pos, listCont2, pos2)</code>	All the elements starting at <code>pos2</code> of <code>listCont2</code> are moved to <code>listCont1</code> before the position specified by the iterator <code>pos</code> .
<code>listCont1.splice(pos, listCont2, beg, end)</code>	All the elements in the range <code>beg...end-1</code> of <code>listCont2</code> are moved to <code>listCont1</code> before the position specified by the iterator <code>pos</code> . Both <code>beg</code> and <code>end</code> are iterators.
<code>listCont.sort()</code>	The elements of <code>listCont</code> are sorted. The sort criterion is <code><</code> .
<code>listCont.sort(oper)</code>	The elements of <code>listCont</code> are sorted. The sort criterion is specified by <code>oper</code> .
<code>listCont1.merge(listCont2)</code>	Suppose that the elements of <code>listCont1</code> and <code>listCont2</code> are sorted. This operation moves all the elements of <code>listCont2</code> into <code>listCont1</code> . After this operation, the elements in <code>listCont1</code> are sorted. Moreover, after this operation, <code>listCont2</code> is empty.
<code>listCont1.merge(listCont2, oper)</code>	Suppose that the elements of <code>listCont1</code> and <code>listCont2</code> are sorted according to the sort criteria <code>oper</code> . This operation moves all the elements of <code>listCont2</code> into <code>listCont1</code> . After this operation, the elements in <code>listCont1</code> are sorted according to the sort criteria <code>oper</code> .
<code>listCont.reverse()</code>	The elements of <code>listCont</code> are reversed.

ตัวอย่างที่ 6

```
//*****  
// Author: D.S. Malik  
//  
// This program illustrates how to use a list container in a  
// program.  
//*****  
  
#include <iostream> //Line 1  
#include <list> //Line 2  
#include <iterator> //Line 3  
#include <algorithm> //Line 4  
using namespace std; //Line 5  
int main() //Line 6  
{ //Line 7  
    list<int> intList1, intList2; //Line 8  
    ostream_iterator<int> screen(cout, " "); //Line 9  
    intList1.push_back(23); //Line 10  
    intList1.push_back(58); //Line 11  
    intList1.push_back(58); //Line 12  
    intList1.push_back(36); //Line 13  
    intList1.push_back(15); //Line 14  
    intList1.push_back(98); //Line 15  
    intList1.push_back(58); //Line 16  
    cout << "Line 17: intList1: "; //Line 17  
    copy(intList1.begin(), intList1.end(), screen); //Line 18  
    cout << endl; //Line 19  
    intList2 = intList1; //Line 20  
    cout << "Line 21: intList2: "; //Line 21  
    copy(intList2.begin(), intList2.end(), screen); //Line 22  
    cout << endl; //Line 23  
    intList1.unique(); //Line 24  
    cout << "Line 25: After removing the consecutive "  
    << "duplicates," << endl  
    << " intList1: "; //Line 25
```

```
copy(intList1.begin(), intList1.end(), screen);           //Line 26
cout << endl;                                           //Line 27
intList2.sort();                                         //Line 28
cout << "Line 29: After sorting, intList2: ";          //Line 29
copy(intList2.begin(), intList2.end(), screen);         //Line 30
cout << endl;                                           //Line 31
return 0;                                               //Line 32
}
```

Sample Run:

Line 17: intList1: 23 58 58 36 15 98 58

Line 21: intList2: 23 58 58 36 15 98 58

Line 25: After removing the consecutive duplicates,

intList1: 23 58 36 15 98 58

Line 29: After sorting, intList2: 15 23 36 58 58 58 98

list::pop_front() & list::pop_back()

- void pop_front ();
- void pop_back ();

จะคล้ายกับ push_front() และ push_back() แต่เป็นการนำ node ออกไปแทน

```
1 // list::pop_front
2 #include <iostream>
3 #include <list>
4 using namespace std;
5
6 int main ()
7 {
8     list<int> mylist;
9     mylist.push_back (100);
10    mylist.push_back (200);
11    mylist.push_back (300);
12    mylist.push_back (400);
13
14    mylist.pop_front();
15    mylist.pop_back ();
```



```
16
17 for (list<int>::iterator it=mylist.begin(); it!=mylist.end(); ++it)
18     cout << *it << " ";
19 return 0;
20 }
```

อ้างอิง: http://www.cplusplus.com/reference/stl/list/pop_front/

Output:

```
200 300
```

list::insert()

insert() เป็นฟังก์ชันที่ใช้ในการแทรก node หรือ กลุ่มของโนด เข้าไป ระหว่างโนดใด ๆ ก็ตาม

- iterator insert (iterator position, const T& x);

parameter แรก นั่นคือ ตำแหน่ง Iterator ส่วน parameter หลังคือ ค่าที่เราจะใส่เข้าไป

- void insert (iterator position, size_type n, const T& x);

จะแตกต่างจาก ฟังก์ชันทางด้านบนเพียงแค่มีกการเพิ่ม size_type n เข้ามาด้วย โดย size_type n จะเป็นตัวกำหนดว่า จะใส่ const T& x เข้าไปเป็นจำนวนกี่ครั้ง

- template <class InputIterator>
void insert (iterator position, InputIterator first, InputIterator last);

เป็นฟังก์ชันที่มีพารามิเตอร์เป็น iterator แทน จึงทำให้สามารถใช้ร่วมกับ STL Container ตัวอื่นได้

```
// inserting into a list
1 #include <iostream>
2 #include <list>
3 #include <vector>
4 using namespace std;
5
6 int main ()
7 {
8     list<int> mylist;
9     list<int>::iterator it;
```

```
10
11 // set some initial values:
12 for (int i=1; i<=5; i++) mylist.push_back(i); // 1 2 3 4 5
13
14 it = mylist.begin();
15 ++it;    // it points now to number 2      ^
16
17 mylist.insert (it,10);                      // 1 10 2 3 4 5
18
19 // "it" still points to number 2            ^
20 mylist.insert (it,2,20);                    // 1 10 20 20 2 3 4 5
21
22 --it;    // it points now to the second 20  ^
23
24 vector<int> myvector (2,30);
25 mylist.insert (it,myvector.begin(),myvector.end());
26                                     // 1 10 20 30 30 20 2 3 4 5
27                                     //      ^
28 cout << "mylist contains:";
29 for (it=mylist.begin(); it!=mylist.end(); it++)
30     cout << " " << *it;
31 cout << endl;
32
33 return 0;
34 }
35
```

อ้างอิง: <http://www.cplusplus.com/reference/stl/list/insert/>

ดังตัวอย่างจะเห็นได้ว่า เราสามารถใช้ ฟังก์ชันที่มีพารามิเตอร์แบบ Iterator มาใช้ร่วมกับ Iterator ของ STL Container ตัวอื่นได้ เช่นในบรรทัดที่ 26 เราได้แทรก vector เข้าไปใน mylist โดยเปลี่ยน vector ให้กลายเป็น list แทน

ตัวอย่างการนำ STL list ไปใช้

การหาจำนวนเฉพาะ ด้วยอัลกอริทึม ตะแกรงกร่อน ของ อีราโทสเทเนส

ตะแกรงกร่อนของอีราโทสเทเนส เป็นอัลกอริทึมที่ให้หาจำนวนเฉพาะจนถึงจำนวนนับ N วิธีการมีดังนี้

1. เขียนจำนวนนับตั้งแต่ 2 จนถึง N ทั้งหมด
2. หาจำนวนที่น้อยที่สุดที่ยังไม่ถูกขีดค่า และเราให้จำนวนนั้นคือ P (P คือจำนวนเฉพาะ)
3. ขีดค่าจำนวนที่เป็นพหุคูณของ P ทุกตัวที่ยังไม่ถูกขีดค่า
4. ถ้ายังคงมีจำนวนนับที่ยังไม่ถูกขีดค่า ให้กลับไปทำขั้นตอนที่ 2

โจทย์

จงเขียนโปรแกรมที่รับจำนวน N จากนั้นหาจำนวนเฉพาะ ที่มีค่าอยู่ระหว่าง 2 - N

ข้อมูลนำเข้า

บรรทัดแรก จำนวนเต็ม N

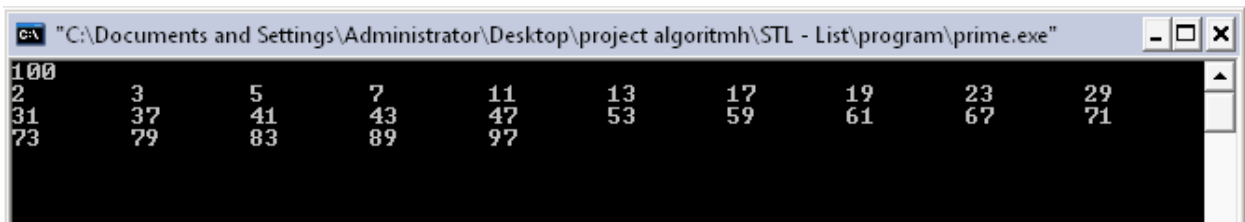
ข้อมูลส่งออก

จำนวนเต็มตั้งแต่ 2 - N

ตัวอย่าง

```
1  #include <iostream>
2  #include <list>
3
4  using namespace std;
5
6  int main(){
7      int n,tmp;
8      list<int> l;
9      list<int>::iterator it,next;
10     cin >> n;
11     for(int i=2;i<=n;i++)
12         l.push_back(i);
13
14     while(!l.empty())
15     {
16         it = l.begin();
17         tmp = *it;
18         while(it!=l.end())
19         {
20             if(*it%tmp==0) {
21                 it = l.erase(it);
22             }
23             else it++;
24         }
```

```
25     cout << tmp << '\t';  
26 }  
27 }
```



The screenshot shows a Windows command prompt window with the following title bar: "C:\Documents and Settings\Administrator\Desktop\project algorithmh\STL - List\program\prime.exe". The window contains the following output:

```
100  
2      3      5      7      11     13     17     19     23     29  
31     37     41     43     47     53     59     61     67     71  
73     79     83     89     97
```

Stack

- Stack คือ โครงสร้างข้อมูลที่มีข้อมูลซึ่งถูกนำเข้ามาที่หลังจะถูกนำมาใช้ก่อน ลักษณะการทำงานแบบนี้เรียกว่า LIFO (Last In First Out)
- ตัวอย่าง Stack ในชีวิตประจำวัน ได้แก่ การวางจานซ้อนกัน โดยจานที่วางที่หลังจะอยู่บน จึงถูกนำมาใช้ก่อน อีกตัวอย่าง คือ เถาปิ่นโต ซึ่งชั้นปิ่นโตที่นำมาวางที่หลัง จะเป็นชั้นที่ถูกนำออกจากเถาก่อน โดยตัวเถาปิ่นโต ก็คือ Stack นั่นเอง
- จะเห็นว่า Stack เป็นโครงสร้างข้อมูลที่มีการเข้าออกได้เพียงหนึ่งทางเท่านั้น
- Stack มีประโยชน์มากในการเขียนโปรแกรม เพราะ Stack จะเก็บลำดับการเรียกฟังก์ชันย่อยในโปรแกรมหลัก รวมทั้งการเรียกตัวเองในฟังก์ชันที่เป็นแบบ Recursive
- เราสามารถสร้าง Stack โดยใช้ Array หรือ Linked List ก็ได้

การใช้ Array สร้าง Stack

- เนื่องจาก Array ต้องมีการกำหนดขนาดที่มากที่สุดที่จะเก็บข้อมูลได้ไว้ล่วงหน้า ดังนั้น Stack ที่สร้างด้วย Array จึงต้องทราบขนาดที่แน่นอนก่อน เช่น ต้องการสร้าง Stack ที่เก็บตัวเลขจำนวนเต็มไม่เกิน **MAX** ตัว (กำหนดให้ **MAX** เป็นค่าคงที่มีค่าเท่ากับ 20) เราสามารถประกาศตัวแปรอาร์เรย์ **stack** ได้ดังนี้

```
int stack[MAX];
```

- การดำเนินการพื้นฐานกับ Stack มีดังนี้
 - การทำ Stack ให้ว่าง (**clear**)
 - การตรวจสอบว่า Stack ว่างหรือไม่ (**empty**)
 - การตรวจสอบว่า Stack เต็มหรือไม่ (**full**)
 - การใส่ข้อมูลใน Stack (**push**)
 - การนำข้อมูลออกจาก Stack (**pop**)
 - การหาจำนวนสมาชิกใน Stack (**size**)
- เราจะใช้ **top** เป็นตัวแปรบอกตำแหน่ง index ของข้อมูลตัวล่าสุดใน Stack ถ้า Stack ว่าง **top** จะมีค่าเท่ากับ -1 และ **top** มีค่าสูงสุดไม่เกิน **MAX - 1**
- การทำ Stack ให้ว่าง (**clear**) จะเป็นการกำหนดให้ **top** มีค่าเท่ากับ -1

```
top = -1;
```

- การตรวจสอบว่า Stack ว่างหรือไม่ (**empty**) จะเป็นการตรวจสอบเงื่อนไขของ

```
top == -1
```

ถ้าเป็น จริง แสดงว่า Stack ว่าง

- การตรวจสอบว่า Stack เต็มหรือไม่ (**full**) จะเป็นการตรวจสอบเงื่อนไขของ

top == MAX - 1

ถ้าเป็น จริง แสดงว่า Stack เต็ม

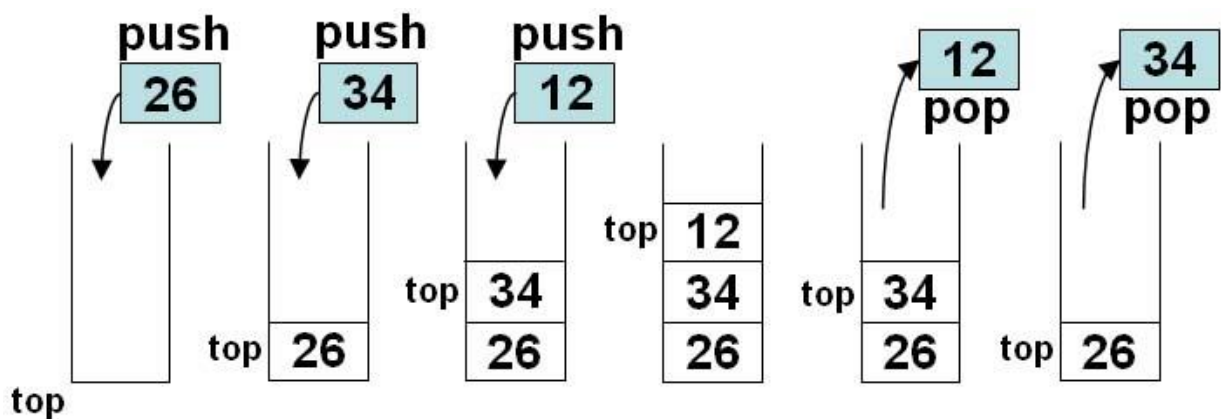
- การใส่ข้อมูลใน Stack (**push**) ต้องแน่ใจว่า Stack ยังไม่เต็ม (**!full**) จากนั้นเพิ่มค่า **top** หนึ่งค่า แล้วนำข้อมูลมาใส่ (สมมุติข้อมูล คือ **num**) ณ ตำแหน่ง **top**

top++; stack[top] = num;

- การนำข้อมูลออกจาก Stack (**pop**) ต้องแน่ใจว่า Stack ไม่ว่าง (**!empty**) จากนั้นเก็บค่าข้อมูลตัวล่าสุดของ Stack ไว้ในตัวแปรตัวหนึ่ง (สมมุติ คือ **num**) แล้วลด **top** หนึ่งค่า จะได้ว่า ค่าที่ถูกนำออกจาก Stack ก็คือ ค่าที่อยู่ในตัวแปร **num** นั้นเอง

num = stack[top]; top--;

- การหาจำนวนสมาชิกใน Stack (**size**) หาได้จาก **top + 1** นั้นเอง



ตัวอย่างที่ 1 สร้าง Stack ด้วย Array ซึ่งจะเก็บค่าตัวเลขจำนวนเต็มลงใน Stack

```
// p01.cpp
#include <iostream>
using namespace std;
#define MAX 20

void display_menu();          void clear( int *tp );
int get_data();              int empty( int t );
int full( int t );           void push( int *sp, int *tp, int item );
int pop( int *sp, int *tp ); void print_stack( int *sp, int t );

void main()
{
    int stack[MAX], top, choice, num;

    display_menu(); cout<<endl<<"Choice : " ; cin>>choice;

    while ( choice ) {
        switch( choice ) {
            case 1: clear( &top ); break;
            case 2: num = get_data(); push( stack, &top, num ); break;
            case 3: num = pop( stack, &top ); break;
            case 4: print_stack( stack, top ); break;
            default: cout<<"Invalid Choice"<<endl; display_menu();
        }
        cout<<endl<<"Choice : " ; cin>>choice;
    }
    cout<<endl<<"# # # End of Program # # #" <<endl;
}

void display_menu()
{
    cout<<endl<<"\nEnter Your Choice"<<endl;
    cout<<endl<<"-----"<<endl ;
    cout<<endl<<"0. Exit"<<endl ;
    cout<<endl<<"1. Clear Stack"<<endl ;
    cout<<endl<<"2. Push Stack"<<endl ;
    cout<<endl<<"3. Pop Stack"<<endl ;
    cout<<endl<<"4. Print Stack"<<endl ;
}

void clear( int *tp )
{
    *tp = -1;
    cout<<"Clear Stack -- Done!!!"<<endl;
}

int get_data()
{
    int item;

    cout<<"Enter a Number : " ; cin>>item ;

    return item;
}

int empty( int t )
{
    return t == -1;
}

int full( int t )
{

```



```
    return t == MAX -1;
}
void push( int *sp, int *tp, int item )
{
    if ( !full( *tp ) ) {
        *tp = *tp + 1;
        sp[*tp] = item;
        cout<<"Push Stack -- Done!!!"<<endl;
    } else
        cout<<"Stack is Full -- Cannot Push!!!"<<endl ;
}

int pop( int *sp, int *tp )
{
    int item = -1;

    if ( !empty( *tp ) ) {
        item = sp[*tp];
        *tp = *tp - 1;
        cout<<"Pop Stack -- Done!!!"<<endl;
    } else
        cout<<"Stack is Empty -- Cannot Pop!!!"<<endl;

    return item;
}

void print_stack( int *sp, int t )
{
    int i;

    cout<< "Stack  : " ;

    if ( !empty( t ) ) {
        for ( i = 0; i <= t; i++ )
            cout<<sp[i]<<endl ;
    } else
        cout<<"Empty"<<endl;
}
```

ตัวอย่างผลลัพธ์จากการรันโปรแกรม (ตัวขีดเส้นใต้ คือ ข้อมูลที่ผู้ใช้ป้อนผ่านแป้นพิมพ์)

01	
02	Enter Your Choice
03	-----
04	0. Exit
05	1. Clear Stack
06	2. Push Stack
07	3. Pop Stack
08	4. Print Stack
09	
10	Choice : <u>1</u>
11	Clear Stack -- Done!!!
12	
13	Choice : <u>4</u>
14	Stack : Empty
15	
16	Choice : <u>2</u>
17	Enter a Number : <u>15</u>
18	Push Stack -- Done!!!
19	
20	Choice : <u>2</u>

```
21 Enter a Number : 24
22 Push Stack -- Done!!!
23
24 Choice : 4
25 Stack : 15 24
26
27 Choice : 2
28 Enter a Number : 36
29 Push Stack -- Done!!!

30
31 Choice : 2
32 Enter a Number : 45
33 Push Stack -- Done!!!
34
35 Choice : 4
36 Stack : 15 24 36 45
37
38 Choice : 3
39 Pop Stack -- Done!!!
40
41 Choice : 4
42 Stack : 15 24 36
43
44 Choice : 3
45 Pop Stack -- Done!!!
46
47 Choice : 4
48 Stack : 15 24
49
50 Choice : 9
51 Invalid Choice
52
53 Enter Your Choice
54 -----
55 0. Exit
56 1. Clear Stack
57 2. Push Stack
58 3. Pop Stack
59 4. Print Stack
60
61 Choice : 1
62 Clear Stack -- Done!!!
63
64 Choice : 4
65 Stack : Empty
66
67 Choice : 2
68 Enter a Number : 62
69 Push Stack -- Done!!!
70
71 Choice : 2
72 Enter a Number : 88
73 Push Stack -- Done!!!
74
75 Choice : 4
76 Stack : 62 88
77
78 Choice : 3
79 Pop Stack -- Done!!!
80
81 Choice : 4
```

82	Stack : 62
83	
84	Choice : <u>0</u>
85	
86	### End of Program ###
87	

Stack STL

การแทนที่โครงสร้างข้อมูลแบบกองซ้อนด้วยแถวลำดับ

แบบลำดับจะต้องมีการจองพื้นที่หน่วยความจำไว้ล่วงหน้าว่าจะมีขนาดเท่าใด โดยแถวลำดับนี้จะปิดปลายด้านหนึ่งไว้ มี 4 ขั้นตอนคือ

- การสร้าง
- การ Push
- การ Pop
- การ แสดง

การนำข้อมูลเข้าไปในกองซ้อน (Push)

เป็นการดำเนินการที่นำข้อมูลเข้าไปเก็บไว้ที่ด้านบนสุดของกองซ้อน (Top of the Stack) เรื่อย ๆ จนกว่ากองซ้อนไม่สามารถนำข้อมูลเข้าไปเก็บได้ จะเรียกว่า กองซ้อนเต็ม (Stack Full)

Example

```
1 // stack::push/pop
2 #include <iostream>
3 #include <stack>
4 using namespace std;
5
6 int main ()
7 {
8     stack<int> mystack;
9
10    for (int i=0; i<5; ++i) mystack.push(i);
11
12    cout << "Popping out elements...";
13    while (!mystack.empty())
14    {
15        cout << " " << mystack.top();
16        mystack.pop();
```

```
17 }  
18 cout << endl;  
19  
20 return 0;  
21 }
```

Output:

Popping out elements... 4 3 2 1 0

การนำข้อมูลออกจากกองซ้อน (Pop)

การทำงานจะตรงข้ามกับ **Push** จะดึงเอาข้อมูลที่อยู่บนสุดออกมาก่อน แต่ก่อนที่จะดึงจะมี
การตรวจสอบว่ากองซ้อนว่างหรือไม่

ถ้าว่างจะไม่สามารถนำข้อมูลออกได้ แสดงว่ากองซ้อนว่าง (Stack Empty)
ถ้าไม่ว่างจะนำเอาข้อมูลออกแล้วเลื่อนตัวชี้ไปยังตำแหน่งถัดลงไป

Example

```
1 // stack::push/pop  
2 #include <iostream>  
3 #include <stack>  
4 using namespace std;  
5  
6 int main ()  
7 {  
8     stack<int> mystack;  
9  
10    for (int i=0; i<5; ++i) mystack.push(i);  
11  
12    cout << "Popping out elements...";  
13    while (!mystack.empty())  
14    {  
15        cout << " " << mystack.top();  
16        mystack.pop();  
17    }  
18    cout << endl;  
19
```

```
20 return 0;
21 }
```

สแตกว่าง (Empty Stack)

นิยาม S.empty ถ้า ()S เป็นสแตก ขบวนการ S.empty) จะส่งผลเป็นจริง ()true เมื่อสแตกว่าง (แต่ส่งผลเป็นเท็จ)false (เมื่อสแตกไม่ว่างหรือสแตกเต็ม การ pop สแตกทุกครั้งจะมีการตรวจสอบข้อมูลในสแตกว่ามีข้อมูลในสแตกหรือไม่ ถ้าไม่มีข้อมูลในสแตก เหลืออยู่ เราก็ไม่สามารถทำการ pop สแตกได้ ในกรณีเช่นนี้เรียกว่า เกิดสถานะ สแตกจม (Stack Underflow) โดยการตรวจสอบว่า สแตกว่างหรือไม่ เราจะตรวจสอบตัวชี้สแตก ว่าเท่ากับ 0 หรือ null หรือไม่ ถ้าเท่ากับ 0 แสดงว่า สแตกว่าง จึงไม่สามารถดึงข้อมูลออกจากสแตกได้

เราสามารถทดสอบว่าสแตกว่างหรือไม่ โดยใช้ฟังก์ชัน isempty ซึ่งจะให้ผลลัพธ์เป็นจริงเมื่อสแตกว่างและเป็นเท็จเมื่อมีข้อมูลในสแตก ดังนี้

```
1 // stack::push/pop
2 #include <iostream>
3 #include <stack>
4 using namespace std;
5
6 int main ()
7 {
8     stack<int> mystack;
9
10    for (int i=0; i<5; ++i) mystack.push(i);
11
12    cout << "Popping out elements...";
13    while (!mystack.empty())
14    {
15        cout << " " << mystack.top();
16        mystack.pop();
17    }
18    cout << endl;
19
20    return 0;
21 }
```

บรรทัดที่ หมายความว่า จะอยู่ในลูปตราบเท่าที่ 13mystack.empty() ไม่ return ค่า true ออกมา

Algorithm การคำนวณแบบ Postfix

ขั้นตอนการแปลง Infix เป็น Postfix

1. พิจารณา Character ใน Infix ที่ละตัว
2. ถ้าพบ วงเล็บเปิด '(' ให้ Push ลง Stack
3. ถ้าพบ Operand (เช่น A, B, C) ให้แสดงออกเป็นผลลัพธ์
4. ถ้าพบ Operator (+, -, *, /, %) ให้พิจารณาดังนี้
 - 4.1 ถ้า Stack ว่าง ให้ Push Operator ลง Stack
 - 4.2 ถ้า Stack ไม่ว่าง ให้เปรียบเทียบค่า Precedence ของ Operator ที่อ่านเข้ามา (op_read) กับ Precedence ของ Operator ณ ตำแหน่ง top ของ Stack (op_top)
 - op_read > op_top ให้ Push op_read ลง Stack
 - op_read <= op_top ให้ Pop ค่าใน Stack ออกเป็นผลลัพธ์มาจนกว่า op_read > op_top จากนั้น ให้ Push op_read ลง Stack
5. ถ้าพบ วงเล็บปิด ')' ให้ Pop ค่าใน Stack ออกเป็นผลลัพธ์จนกว่าจะพบ วงเล็บเปิด '(' โดยที่ไม่ต้องแสดง วงเล็บเปิด '(' และ วงเล็บปิด ')' ในผลลัพธ์
6. เมื่อพิจารณา Character จนหมด Infix แล้ว ให้ Pop ค่าใน Stack ออกเป็นผลลัพธ์จน Stack ว่าง

สิ่งที่เห็นข้างล่าง คือ ตัวอย่างผลลัพธ์จากการรันโปรแกรม โดยที่ ส่วนที่ขีดเส้นใต้ คือ สิ่งที่ใช้ป้อนลงไปผ่าน แป้นพิมพ์ ซึ่งจะพบว่า โปรแกรมจะแสดงผลไปเรื่อยๆ จนกว่าผู้ต้องการออกจากโปรแกรม

ความสำคัญของตัวดำเนินการ

เครื่องหมาย	ความสำคัญเมื่ออยู่ในStack	ความสำคัญเมื่ออยู่ที่อินพุต
** (ยกกำลัง)	3	4
*, /	2	2
+, -	1	1

(0	3
---	---	---

A + B * C

เครื่องหมาย	Stack	นิพจน์ Postfix
A	ว่าง	A
+	+	A
B	+	AB
*	+, *	AB
C	+, *	ABC
		ABC*+

ตัวอย่างโปรแกรม

```
#include <cstdlib>
#include <iostream>
#include <string>
#include <stack>
using namespace std;
bool isOperator(char a);
void putter(stack<char> &st,char c);
bool morethanequal(char a,char b);
bool underequal(char a,char b);
bool morethan(char a,char b);
string in,out;
int main(int argc, char *argv[])
{
    stack<char> st;
    getline(cin,in);
    for(int i=0,len=in.size();i<len;i++)
    {
        if(isOperator(in[i]))
        {
            putter(st,in[i]);
        }
        else {out += in[i]; out += ' ';}
    }
}
```

```
}
while(!st.empty()) {out += st.top(); out += ' ';st.pop();}
cout << endl << out << endl;
system("PAUSE");
return EXIT_SUCCESS;
}

void putter(stack<char> &st,char c)
{
    if(st.empty())
    {
        st.push(c);
    }
    else // stack is not empty
    {
        if(c=='(') st.push(c);
        else if(c==')'){
            while(!st.empty())
            {
                if(st.top()=='('){ st.pop();break;}
                out += st.top();
                out += ' ';
                st.pop();
            }
        }
        else
        {
            while(!st.empty()&&morethanequal(st.top(),c)&&st.top()!='(')
            {
                out += st.top();
                out += ' ';
                st.pop();
            }
            st.push(c);
        }
    }
}
```



```
    }  
}
```

```
bool morethanequal(char a,char b)
```

```
{  
    int aa,bb;  
    if(a=='+')aa=1;  
    else if(a=='-')aa=1;  
    else if(a=='*')aa=2;  
    else if(a=='/')aa=2;  
    else if(a=='(')aa=3;  
    if(b=='+')bb=1;  
    else if(b=='-')bb=1;  
    else if(b=='*')bb=2;  
    else if(b=='/')bb=2;  
    else if(b=='(')aa=3;  
    return aa>=bb;  
}
```

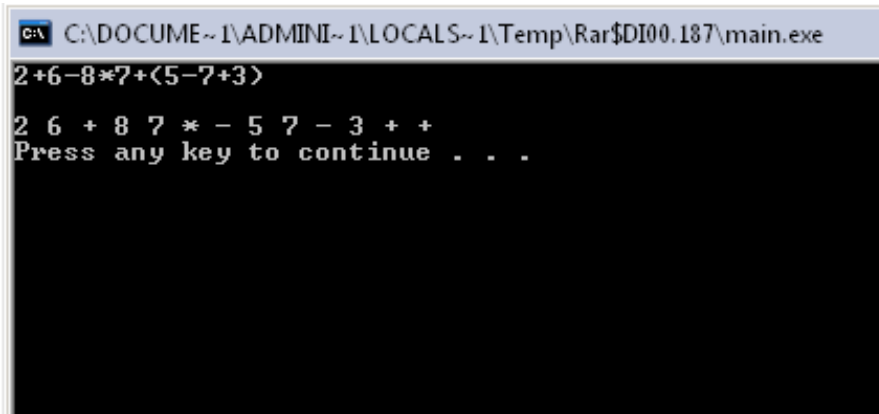
```
bool morethan(char a,char b)
```

```
{  
    int aa,bb;  
    if(a=='+')aa=1;  
    else if(a=='-')aa=1;  
    else if(a=='*')aa=2;  
    else if(a=='/')aa=2;  
    else if(a=='(')aa=3;  
    if(b=='+')bb=1;  
    else if(b=='-')bb=1;  
    else if(b=='*')bb=2;  
    else if(b=='/')bb=2;  
    else if(b=='(')aa=3;  
    return aa>bb;  
}
```

```
bool isOperator(char a)
```

```
{
```

```
return (a=='*'||a=='-'||a=='/'||a=='+'||a=='('||a==')');  
}
```



The screenshot shows a Windows command prompt window with the title bar "C:\ C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\Rar\$DI00.187\main.exe". The command prompt displays the expression `2+6-8*7+(5-7+3)` on the first line. On the second line, the characters `2 6 + 8 7 * - 5 7 - 3 + +` are displayed. On the third line, the text `Press any key to continue . . .` is shown.

Queue

- Queue คือ โครงสร้างข้อมูลที่มีข้อมูลใหม่ จะนำเข้าทาง **rear** และเมื่อมีการนำข้อมูลออก จะนำออกทาง **front** โดยที่ข้อมูลที่เข้า Queue ก่อน จะเป็นข้อมูลที่ออกจาก Queue ก่อน ลักษณะการทำงานแบบนี้ เรียกว่า FIFO (First In First Out)
- ตัวอย่าง Queue ในชีวิตประจำวัน ได้แก่ การเข้าแถวซื้อตั๋วหนัง คนที่มาก่อนจะได้ซื้อตั๋วก่อน เป็นต้น
- จะเห็นว่า Queue เป็นโครงสร้างข้อมูลที่มีการเข้าออก 2 ทาง (นั่นคือ เข้าทาง **rear** ออกทาง **front**)
- Queue ได้ถูกนำมาใช้ในระบบงานคอมพิวเตอร์ เช่น การเข้า Queue เพื่อรอการประมวลผล และ การเข้า Queue รอพิมพ์ผลลัพธ์จากเครื่องพิมพ์ เป็นต้น
- เราสามารถสร้าง Queue โดยใช้ Array หรือ Linked List ก็ได้

การใช้ Array สร้าง Queue




- เนื่องจาก Array ต้องมีการกำหนดขนาดที่มากที่สุดที่จะเก็บข้อมูลได้ไว้ล่วงหน้า ดังนั้น Queue ที่สร้างด้วย Array จึงต้องทราบขนาดที่แน่นอนก่อน เช่น

อุโมงค์รถแห่งหนึ่ง สามารถเก็บรถไว้ในอุโมงค์ได้เต็มที่ **MAX** คัน (กำหนดให้ **MAX** เป็นค่าคงที่มีค่าเท่ากับ 5) โดยที่ รถที่ถูกนำเข้าอุโมงค์ก่อน จะได้รับบริการซ่อมก่อน และเมื่อซ่อมรถคันใดเสร็จแล้ว ก็จะนำรถคันนั้นออกจากอุโมงค์ไป จะเห็นว่า อุโมงค์นี้ใช้หลักการบริการแบบ Queue

กำหนดให้ **que** เป็นตัวแปรอาร์เรย์ ขนาด **MAX** โดยมีชนิดข้อมูลของอาร์เรย์แต่ละช่องเป็น **CAR** ดังนั้น จะมีการประกาศตัวแปรดังนี้

CAR que [MAX] ;

- เราจะใช้ **front** เป็นตัวแปรบอกตำแหน่ง index ของข้อมูลตัวแรกสุดของ Queue และ จะใช้ **rear** เป็นตัวแปรบอกตำแหน่ง index ของข้อมูลตัวล่าสุดใน Queue

0	1	2	3	4	front	rear
					0	2

0	1	2	3	4	front	rear
					2	3

0	1	2	3	4	front	rear
---	---	---	---	---	-------	------

					3	1
---	---	--	---	--	---	---

- การดำเนินการพื้นฐานกับ Queue มีดังนี้
 - การทำ Queue ให้ว่าง (**clearq**)
 - การตรวจสอบว่า Queue ว่างหรือไม่ (**emptyq**)
 - การตรวจสอบว่า Queue เต็มหรือไม่ (**fullq**)
 - การใส่ข้อมูลลงใน Queue (**enq**)
 - การนำข้อมูลออกจาก Queue (**deq**)
 - การหาจำนวนสมาชิกที่อยู่ใน Queue (**length**)

- A. การทำ Queue ให้ว่าง (**clearq**) จะเป็นการกำหนดให้ **front** และ **rear** มีค่าเท่ากับ -1

front = rear = -1;

0	1	2	3	4	front	rear
					-1	-1

- B. การตรวจสอบว่า Queue ว่างหรือไม่ (**emptyq**) จะเป็นการตรวจสอบเงื่อนไขของ

front == -1

ถ้าเป็น จริง แสดงว่า Queue ว่าง

- C. การตรวจสอบว่า Queue เต็มหรือไม่ (**fullq**) จะเป็นการตรวจสอบเงื่อนไขของ

front == (rear + 1) % MAX

ถ้าเป็น จริง แสดงว่า Queue เต็ม

0	1	2	3	4	front	rear
					0	4

0	1	2	3	4	front	rear
					2	1

0	1	2	3	4	front	rear
					3	2

D. การใส่ข้อมูลลงใน Queue (**enq**) สมมุติข้อมูลที่ใส่ คือ **item**


- ต้องแน่ใจว่า Queue ไม่เต็ม (**!fullq**)
- เมื่อ Queue ไม่เต็ม จะแบ่งได้อีก 2 กรณี คือ
 - Queue ว่าง (**emptyq**) ให้กำหนด **front = rear = 0;**
 - Queue ไม่ว่าง (**!emptyq**) ให้กำหนด **rear = (rear + 1) % MAX;**
- que[rear] = item;**

D1 -- ตัวอย่างการนำข้อมูลเข้าไปใน Queue ที่ว่าง

ก่อนมีการนำรถ **Mitsubishi** เข้าอยู่



0	1	2	3	4	front	rear
					-1	-1

หลังนำรถ **Mitsubishi** เข้าอยู่

0	1	2	3	4	front	rear
					0	0

D2 -- ตัวอย่างการนำข้อมูลเข้าไปใน Queue ที่ไม่ว่าง แต่ไม่เต็ม

ก่อนมีการนำรถ **Nissan** เข้าอยู่

0	1	2	3	4	front	rear
					1	2

หลังนำรถ **Nissan** เข้าอยู่

0	1	2	3	4	front	rear
---	---	---	---	---	-------	------

					1	3
--	---	---	---	--	---	---

D3 -- อีกตัวอย่างของการนำข้อมูลเข้าไปใน Queue ที่ไม่ว่าง แต่ไม่เต็ม

ก่อนมีการนำรถ **Benz** เข้าอยู่

0	1	2	3	4	front	rear
					2	4

หลังนำรถ **Benz** เข้าอยู่

0	1	2	3	4	front	rear
					2	0

D4 -- ตัวอย่างการนำข้อมูลเข้าไปใน Queue ที่เต็ม

ก่อนมีการนำรถ **Ford** เข้าอยู่

0	1	2	3	4	front	rear
					2	1

ไม่สามารถนำรถ **Ford** เข้าอยู่ได้

0	1	2	3	4	front	rear
					2	1

E. การนำข้อมูลออกจาก Queue (`deq`)

1. ต้องแน่ใจว่า Queue ไม่ว่าง (`!empty`)
2. กำหนดค่าข้อมูลที่จะนำออกให้กับ item `item = que[front];`
3. เมื่อ Queue ไม่ว่าง จะแบ่งได้อีก 2 กรณี
 - 3.1 เหลือข้อมูลตัวเดียว ให้ทำ Queue ให้ว่าง (`clearq`)
 - 3.2 เหลือข้อมูลมากกว่าหนึ่งตัว ให้กำหนด `front = (front + 1) % MAX;`
4. คืนค่า `item`

E1 -- ตัวอย่างการนำข้อมูลออกจาก Queue แต่มีข้อมูลเหลืออยู่ตัวเดียว

ก่อนมีการนำรถออกจากแถว

0	1	2	3	4	front	rear
					3	3

นำรถ **Mazda** ออกจากแถว

0	1	2	3	4	front	rear
					-1	-1

E2 -- ตัวอย่างการนำข้อมูลออกจาก Queue โดยมีข้อมูลเหลือมากกว่าหนึ่งตัว






ก่อนมีการนำรถออกจากแถว

0	1	2	3	4	front	rear
					1	3

นำรถ **BMW** ออกจากแถว

0	1	2	3	4	front	rear
					2	3

E3 -- อีกตัวอย่างของการนำข้อมูลออกจาก Queue โดยมีข้อมูลเหลือมากกว่าหนึ่งตัว

ก่อนมีการนำรถออกจากอยู่						
0	1	2	3	4	front	rear
					4	1
นำรถ Mazda ออกจากอยู่						
0	1	2	3	4	front	rear
					0	1

E4 -- ตัวอย่างการนำข้อมูลออกจาก Queue ที่ว่าง

ก่อนมีการนำรถออกจากอยู่						
0	1	2	3	4	front	rear
					-1	-1
ไม่มีรถที่นำออกจากจากอยู่ได้						
0	1	2	3	4	front	rear
					-1	-1

F การหาจำนวนสมาชิกที่อยู่ใน Queue (**length**)

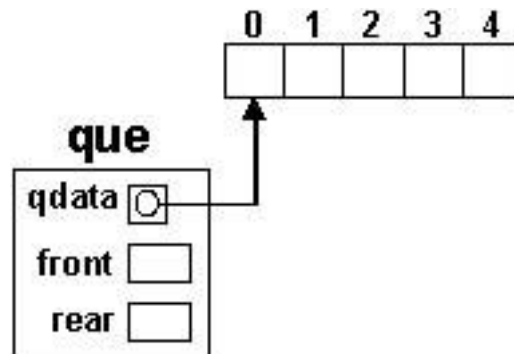
- ถ้า **rear** \geq **front** จำนวนสมาชิกหาได้จาก **rear - front + 1**
- ถ้า **rear** $<$ **front** จำนวนสมาชิกหาได้จาก **MAX + rear - front + 1**

F1 -- ตัวอย่าง Queue ที่มีจำนวนสมาชิก 3 ตัว

0	1	2	3	4	front	rear
					1	3

F2 -- ตัวอย่าง Queue ที่มีจำนวนสมาชิก 4 ตัว

0	1	2	3	4	front	rear
					2	0



ตัวอย่างที่ 4 สร้าง Queue ด้วย Array ซึ่งจะเก็บค่าตัวเลขจำนวนเต็มลงใน Queue

```
// p04.c
#include <iostream>
using namespace std;
#define MAX 5

struct QUEUE { int qdata[MAX]; int front; int rear; };

typedef struct QUEUE QTYPE; typedef QTYPE *QPTR;

void display_menu();          void clearq( QPTR qp );
int  emptyq( QTYPE qvar );    int  fullq( QTYPE qvar );
void enq( QPTR qp, int item ); int  deq( QPTR qp );
void printq( QTYPE qvar );

void main()
{
    QTYPE que; int choice, num;

    display_menu(); cout<<endl<<"Choice : " ; cin>>choice;

    while ( choice ) {
        switch ( choice ) {
            case 1: clearq( &que ); break;
            case 2: cout<<"Enter Number : " ; cin>>num ;
                    enq( &que, num ); break;
            case 3: num = deq( &que );
                    if ( num != -1 )
                        cout<<"Dequeue %d Done!!!"<< num ;
                    break;
            case 4: printq( que ); break;
            default: cout<<"Invalid Choice"<<endl; display_menu();
        }
        cout<<endl<<"Choice : " ; cin>>choice;
    }
}
```

```
    }
    cout<<endl<<"* * * End of Program * * *"<<endl;
}

void display_menu()
{
    cout<<endl<<"QUEUE MENU"<<endl ;
    cout<<"-----"<<endl;
    cout<<"0.Exit 1.Clear 2.Enqueue 3.Dequeue 4.Print"<<endl;
}

void clearq( QPTR qp )
{
    qp->front = qp->rear = -1; cout<<"Clear Done!!!"<<endl;
}

int emptyq( QTYPE qvar )
{
    return qvar.front == -1;
}

int fullq( QTYPE qvar )
{
    return qvar.front == ( qvar.rear + 1 ) % MAX;
}

void enq( QPTR qp, int item )
{
    if ( fullq( *qp ) )
        cout<<"Queue Overflow!!!"<<endl;
    else {
        if ( emptyq( *qp ) )
            qp->front = qp->rear = 0;
        else
            qp->rear = ( qp->rear + 1 ) % MAX;
        qp->qdata[qp->rear] = item;
        cout<<"Enqueue %d Done!!!"<< item <<endl;
    }
}

int deq( QPTR qp )
{
    int item = -1;

    if ( emptyq( *qp ) )
        cout<<"Queue Underflow!!!"<<endl;
    else {
        item = qp->qdata[qp->front];
        if ( qp->front == qp->rear )
            clearq( qp );
        else
            qp->front = ( qp->front + 1 ) % MAX;
    }
    return item;
}

void printq( QTYPE qvar )
{
    int fx = qvar.front, rx = qvar.rear;
```

```
cout<<"QUEUE : "<<endl;

if ( emptyq( qvar ) )
    cout<<"Empty" <<endl;
else {
    if ( fx <= rx )
        while ( fx <= rx ) {
            cout<<qvar.qdata[fx] ; fx++;
        }
    else {
        while ( fx <= MAX - 1 ) {
            cout<<qvar.qdata[fx] ; fx++;
        }
        fx = 0;
        while ( fx <= rx ) {
            cout<< qvar.qdata[fx];  fx++;
        }
    }
    cout<<endl;
}
}
```

ตัวอย่างผลลัพธ์จากการรันโปรแกรม (ตัวขีดเส้นใต้ คือ ข้อมูลที่ผู้ใช้ป้อนผ่านแป้นพิมพ์)

```
01
02 QUEUE MENU
03 -----
04 0.Exit 1.Clear 2.Enqueue 3.Dequeue 4.Print
05
06 Choice : 1
07 Clear Done!!!
08
09 Choice : 3
10 Queue Underflow!!!
11
12 Choice : 4
13 QUEUE : Empty
14
15 Choice : 2
16 Enter Number : 14
17 Enqueue 14 Done!!!
18
19 Choice : 2
20 Enter Number : 25
21 Enqueue 25 Done!!!
22
23 Choice : 4
24 QUEUE : 14 25
25
26 Choice : 2
27 Enter Number : 36
28 Enqueue 36 Done!!!
29
30 Choice : 2
31 Enter Number : 47
32 Enqueue 47 Done!!!
33
34 Choice : 2
35 Enter Number : 58
36 Enqueue 58 Done!!!
37
38 Choice : 4
39 QUEUE : 14 25 36 47 58
40
41 Choice : 2
42 Enter Number : 69
43 Queue Overflow!!!
44
45 Choice : 4
46 QUEUE : 14 25 36 47 58
47
48 Choice : 9
49 Invalid Choice
50
51 QUEUE MENU
52 -----
53 0.Exit 1.Clear 2.Enqueue 3.Dequeue 4.Print
54
55 Choice : 4
56 QUEUE : 14 25 36 47 58
57
58 Choice : 2
59 Enter Number : 69
```

```
60 Queue Overflow!!!
61
62 Choice : 4
63 QUEUE : 14 25 36 47 58
64
65 Choice : 3
66 Dequeue 14 Done!!!
67
68 Choice : 3
69 Dequeue 25 Done!!!
70
71 Choice : 4
72 QUEUE : 36 47 58
73
74 Choice : 3
75 Dequeue 36 Done!!!
76
77 Choice : 3
78 Dequeue 47 Done!!!
79
80 Choice : 3
81 Clear Done!!!
82 Dequeue 58 Done!!!
83
84 Choice : 4
85 QUEUE : Empty
86
87 Choice : 3
88 Queue Underflow!!!
89
90 Choice : 4
91 QUEUE : Empty
92
93 Choice : 2
94 Enter Number : 37
95 Enqueue 37 Done!!!
96
97 Choice : 0
98
99 * * * End of Program * * *
```

การใช้ STL กับ Queue

การนำข้อมูลเข้า

การจะเพิ่มข้อมูลเข้าไปในคิว จะกระทำที่ตำแหน่ง REAR หรือท้ายคิว และก่อนที่จะเพิ่มข้อมูลจะต้องตรวจสอบก่อนว่าคิวเต็มหรือไม่ โดยการเปรียบเทียบค่า REAR ว่า เท่ากับค่า MAX QUEUE หรือไม่ หากว่าค่า REAR = MAX QUEUE แสดงว่าคิวเต็มไม่สามารถเพิ่มข้อมูลได้ แต่หากไม่เท่า แสดงว่าคียังมีที่ว่างสามารถเพิ่มข้อมูลได้ เมื่อเพิ่มข้อมูลเข้าไปแล้ว ค่า REAR ก็จะเป็นค่าตำแหน่งท้ายคิวใหม่

Example

```
1 // queue::push/pop
2 #include <iostream>
3 #include <queue>
4 using namespace std;
5
6 int main ()
7 {
8     queue<int> myqueue;
9     int myint;
10
11     cout << "Please enter some integers (enter 0 to end):\n";
12     do {
13         cin >> myint;
14         myqueue.push (myint);
15     } while (myint);
16
17     cout << "myqueue contains: ";
18     while (!myqueue.empty())
19     {
20         cout << " " << myqueue.front();
21         myqueue.pop();
22     }
```

```
23
24  return 0;
25 }
26
```

การนำข้อมูลออก

การนำข้อมูลออกจากคิวจะกระทำที่ตำแหน่ง FRONT หรือส่วนที่เป็นหัวของคิว โดยก่อนที่จะนำข้อมูลออกจากคิวจะต้องมีการตรวจสอบก่อนว่ามีข้อมูลอยู่ในคิว หรือไม่ หากไม่มีข้อมูลในคิวหรือว่าคิวว่าง ก็ไม่สามารถนำข้อมูลออกจากคิวได้

Example

```
1  // queue::push/pop
2  #include <iostream>
3  #include <queue>
4  using namespace std;
5
6  int main ()
7  {
8      queue<int> myqueue;
9      int myint;
10
11     cout << "Please enter some integers (enter 0 to end):\n";
12
13     do {
14         cin >> myint;
15         myqueue.push (myint);
16     } while (myint);
17
18     cout << "myqueue contains: ";
19     while (!myqueue.empty())
20     {
21         cout << " " << myqueue.front();
22         myqueue.pop();
23     }
```

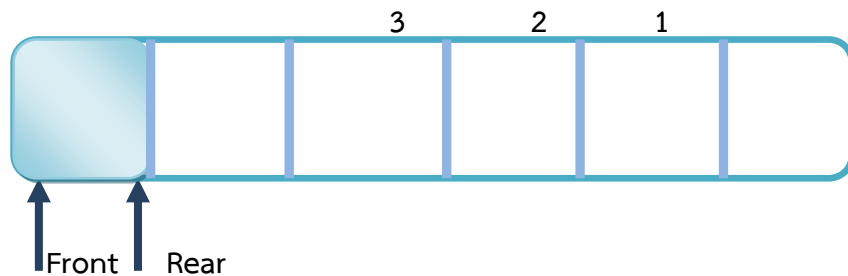


```
24  
25 return 0;  
26 }
```

คิวแบบวงกลม (Circular Queue)

คิวแบบวงกลมจะมีลักษณะเหมือนคิวธรรมดา คือ มีตัวชี้ FRONT และ REAR ที่แสดงตำแหน่งหัวคิวและท้ายคิวตามลำดับ โดย FRONT และ REAR จะมีการเลื่อนลำดับทุกครั้งเมื่อมีการนำข้อมูลเข้าและออกจากคิว แต่จะแตกต่างจากคิวธรรมดาคือ คิวธรรมดาเมื่อ REAR ชี้ที่ตำแหน่งสุดท้ายของคิว จะทำให้เพิ่มข้อมูลเข้าในคิวอีกเมื่อไม่ได้ เนื่องจาก ค่า REAR=MAX QUEUE ซึ่งแสดงว่าคิวนั้นเต็ม ไม่สามารถเพิ่มข้อมูลเข้าไปได้อีก ทั้งๆ ที่ยังมีเนื้อที่ของคิวเหลืออยู่ก็ตาม ทำให้การใช้เนื้อที่ของคิวไม่มีประสิทธิภาพ

Max Queue



จากรูป แสดงคิวที่ค่า REAR ชี้ที่ตำแหน่งสุดท้ายของคิว ทำให้ไม่สามารถนำข้อมูลเข้าได้อีก

สำหรับคิวแบบวงกลม จะมีวิธีจัดการกับปัญหานี้ คือ เมื่อมีข้อมูลเพิ่มเข้ามาในคิว ในลักษณะดังกล่าว คือ ขณะที่ REAR ชี้ตำแหน่งสุดท้ายของคิว ถ้าหากมีการเพิ่มค่าของ REAR REAR จะสามารถวนกลับมาชี้ยังตำแหน่งแรกของคิวได้ ซึ่งจะทำให้คิวมีลักษณะเป็นแบบวงกลม

การนำฟังก์ชัน enqueue และ dequeue มาใช้

โดยปกติแล้ว STL queue นั้น ไม่มีฟังก์ชัน enqueue และ dequeue มาให้ ฉะนั้นหากเราต้องการใช้ ฟังก์ชัน enqueue หรือ dequeue นั้น เราต้องทำขึ้นมาใช้เอง โดยอาจจะใช้การ inheritance มาจาก class queue เดิม ก็ได้ เช่น

```
template<class T>  
class Queue : public queue<T> {  
public:  
    T dequeue() {  
        T tmp = queue<T>::front();  
        queue<T>::pop();  
        return tmp;  
    }  
    void enqueue(const T& el) {  
        push(el);  
    }  
};
```

```
}  
};
```

ซึ่งต่อไปนี้ เราอาจใช้ฟังก์ชันที่เราสืบทอดมา ใช้ในการเขียนโปรแกรมได้ตลอด แทนฟังก์ชัน queue เลยก็ได้ เพราะเหมือนกันทุกอย่าง เพียงแต่เพิ่มฟังก์ชัน ทั้ง นี้มาเท่านั้น 2

ตัวอย่างการใช้งาน

Circular Queue in c	Circular Queue in c++
<pre>//Program for circular queue #include<stdio.h> #include<conio.h> #define max 5 void insert(); void del(); void display(); int cq[max]; int front=-1,rear=-1; int main() { int choice; while(1) { printf("\nPress 1 to Enqueue\n"); printf("Press 2 to Dequeue\n"); printf("Press 3 to Display\n");</pre>	<pre>#include<iostream> #include<cstdlib> #define MAX_SIZE 4 using namespace std; class Queue{ private: int item[MAX_SIZE]; int head; int tail; public: Queue(); void enqueue(int); int dequeue(); int size(); void display(); bool isEmpty(); bool isFull(); }; Queue::Queue(){ head = 0; tail = 0; } void Queue::enqueue(int data){</pre>

```
printf("Press 4 to Exit\n");
printf("Enter your choice:");
scanf("%d",&choice);
switch(choice)
{
    case 1:
        insert();
        break;
    case 2:
        del();
        break;
    case 3:
        display();
        break;
    case 4:
        return 0;
    default:
        printf("\n
Invalid Choice:");
        break;
}

getch();
}
```

```
item[tail] = data;
tail = (tail+1)%MAX_SIZE;
}

int Queue::dequeue(){
    int temp;
    temp = item[head];
    head = (head+1)%MAX_SIZE;
    return temp;
}

int Queue::size(){
    return (tail - head);
}

void Queue::display(){
    int i;
    if(!this->isEmpty()){
        for(i=head; i!=tail;
i=(i+1)%MAX_SIZE){
            cout<<item[i]<<endl;
        }
    }else{
        cout<<"Queue Underflow"<<endl;
    }
}

bool Queue::isEmpty(){
    if(abs(head == tail)){
        return true;
    }else{
```

```
void insert()
{
    int add_item;

    if((front==0 && rear==max-1)||
(front==rear+1))
    {
        printf("\n Queue
Overflow.\n\n");

        return;
    }

    if(front==max-1)
    {
        front=0;

        rear=0;
    }
    else
    {
        if(rear==max-1)

            rear=0;

        else

            rear=rear+1;

        getch();
    }

    printf("\n Input the element for
insertion in queue:");

    scanf("%d",&add_item);
```

```
        return false;
    }
}

bool Queue::isFull(){
    if(head==(tail+1)%MAX_SIZE){
        return true;
    }else{
        return false;
    }
}

int main(){
    Queue queue;
    int choice, data;
    while(1){
        cout<<"\n1. Enqueue\n2.
Dequeue\n3. Size\n4. Display all
element\n5. Quit";

        cout<<"\nEnter your choice: ";
        cin>>choice;
        switch(choice){
            case 1:
                if(!queue.isFull()){
                    cout<<"\nEnter data: ";
                    cin>>data;
                    queue.enqueue(data);
                }else{
                    cout<<"Queue is Full"<<endl;
                }
                break;
            case 2:
                if(!queue.isEmpty()){
```

```
        cq[rear]=add_item;
        printf("Element inserted.");
    }

void del()
{
    if(front== -1)
    {
        printf("\n Queue
Underflow.\n\n");
        return;
    }

    printf("\n Element deleted from
the Queue is %d \n\n",cq[front]);
    if(front==rear)
    {
        front=-1;
        rear=-1;
    }
    else
    {
        if(front==max-1)
        {
            front=0;
        }
    }
}
```

```
        cout<<"The data dequeued
is : "<<queue.dequeue();
    }else{
        cout<<"Queue is
Empty"<<endl;
    }
    break;
    case 3:
        cout<<"Size of Queue is
"<<queue.size();
        break;
    case 4:
        queue.display();
        break;
    case 5:
        exit(0);
        break;
    }
}
return 0;
}
```

```
        else

            front=front+1;

        }

    }

void display()

{

    int frontpos=front;

    int rearpos= rear;

    if(front== -1)

    {

        printf("\n Queue is

empty.\n");

        return;

    }

    printf("\n Queue elements is:

\n");

    if(frontpos<=rearpos)

    {

while(frontpos<=rearpos)

        {

printf("%d ",cq[frontpos]);

frontpos++;

}
```

```
        }

    }

    else
    {
        while(frontpos<=max-1)
        {
            printf("%d\n",cq[frontpos]);
            frontpos++;
        }
        frontpos=0;
        while(frontpos<=rearpos)
        {
            printf("%d\n",cq[frontpos]);
            frontpos++;
        }
    }

    printf("\n");
}
```

คิวลำดับความสำคัญ (priority queue)

คิวอาจมีการจัดลำดับความสำคัญของสมาชิกจึงมีการสร้างเป็นคิวลำดับความสำคัญ (Priority Queue) เช่นในระบบปฏิบัติการจะให้โปรเซสที่มีความสำคัญที่สุดทำงานก่อน แต่เนื่องจากโปรเซสที่เข้ามาไม่ได้เรียงตามความสำคัญ ซึ่งการจัดลำดับมี 2 แบบ คือ ให้ความสำคัญ จากค่าน้อยไปหาค่ามาก (Ascending) และจากค่ามากไปหาค่าน้อย (Descending) โดยส่วนใหญ่ใช้ค่ามากมีความสำคัญ มากกว่า ในการเพิ่มสมาชิกเข้ามาในคิวไม่จำ เป็นต้องเข้ามาตามลำดับความสำคัญอาจสลับไปมาได้แต่การนำออกมาจากคิวจะต้องตามลำดับความสำคัญ ดังนั้นในการลบ

สมาชิกออกจากคิวจึงต้องมีการทำงาน 2 เรื่อง คือ ต้องหาสมาชิกที่มีความสำคัญมากที่สุด ทำให้ต้องเข้าไปเรียกใช้งานสมาชิกทุกตัว และเมื่อลบสมาชิกออกจากคิวในช่วงกลางจะต้องทำการขยับสมาชิกตัวถัดไปมาอยู่แทนในการแก้ไขปัญหานี้สองเรื่องมีอยู่หลายวิธีเช่นกำหนดตัวแปรให้ไปย้ตำแหน่งสมาชิกที่สำคัญที่สุด หลังจากที่ถูกลบออกไปเป็นตำแหน่งทำให้ไม่ต้องขยับสมาชิกตัวอื่นเมื่อเพิ่มสมาชิกใหม่ก็เก็บไว้ที่ตำแหน่งนี้แทนแต่มีข้อเสียก็คือต้องค้นหาสมาชิกที่สำคัญที่สุดทุกครั้งที่มีการลบเช่นเดิม การทำงานแบบนี้จะทำ เมื่อมีการลบสมาชิกออกจากคิวแต่มีวิธีที่เหมาะสมกว่าคือการจัดให้สมาชิกทุกตัวเรียงในคิวตามลำดับ โดยสมาชิกในตำแหน่ง Front มีความสำคัญสูงสุดและลดหลั่นลงมาจนถึงตำแหน่ง Rear ที่มีความสำคัญน้อยที่สุดการทำงานจะทำเมื่อมีการเพิ่มสมาชิกใหม่เข้ามาในคิวส่วนการลบสมาชิกในคิวไม่ต้องทำงานเพิ่มเติมดังที่ผ่านมาเพื่อหาสมาชิกที่สำคัญที่สุดเพราะสมาชิกที่ Front จะสำคัญที่สุดสามารถลบออกไปได้ทันที เมื่อใดที่มีสมาชิกใหม่เพิ่มเข้ามาจะทำการจัดเรียงตามลำดับโดยการหาตำแหน่งที่ถูกต้องให้กับสมาชิก

MAX Priority Queue	MIN Priority Queue
<pre> #include<iostream> #include<queue> using namespace std; int main(){ priority_queue<float> q; q.push(44.5); q.push(5.4); q.push(66.6); cout<<q.top()<<' '; q.pop(); cout<<q.top()<< endl; q.pop(); q.push(11.11); q.push(55.5); q.push(33.33); q.pop(); while(!q.empty()){ cout<<q.top()<<' '; q.pop(); } cout<<endl; return 0; } </pre>	<pre> #include <iostream> #include <queue> using namespace std; struct compare { bool operator()(const int& l, const int& r) { return l > r; } }; int main() { priority_queue<int,vector<int>, compare > pq; pq.push(3); pq.push(5); pq.push(1); pq.push(8); while (!pq.empty()) { cout << pq.top() << endl; pq.pop(); } } </pre>

	<pre> } cin.get(); } </pre>
Link List Queue	Link List Queue Using STL
<pre> #include <iostream> #include <conio.h> using namespace std; typedef struct ListNode{ int data; // the element data struct ListNode *next; // next link }ListElem; //----- void Push (int); ListElem *Pop();//Pop item from the Queue void printall();//print out all items on the screen int countitem();//return the number of items in the Queue ListElem *find(int); ListElem *findmin(); ListElem *findmax(); ListElem *pfirst; ListElem *plast; </pre>	<pre> #include<iostream> #include<list> using namespace std; int main(){ list <int> ql; char choice; int i; cout<<"do you want to run this job ? y/n "; cin>>choice; while(choice=='y'){ cin>>i; ql.push_back(i); ql.sort(); ql.reverse(); cout<<"do you want to run this job ? y/n "; cin>>choice; } for (list<int>::iterator it=ql.begin(); it!=ql.end(); ++it){ cout<<*it<<endl; </pre>

```
//Add an item to the Queue
void Push (int val)
{

    int t;
    ListElem *item;//new element to be Pushed
    item=new ListElem; //allocate space
    if(!item) {cout<<"Memory problem..."<<endl; }

    item->data=val;

//Push a new item to the empty Queue

    if(pfirst==NULL && plast==NULL){
//The first and last item point to the new item when
they are null--empty stack.
        item->next=NULL;
        pfirst=item;
        plast=item;

        cout<<"Pushed:"<<item->data<<endl;
    }

//Push a new item at the beginning of the stack
else
{

    item->next=pfirst;
    pfirst=item;
    cout<<"Pushed"<<item->data<<endl;
    }

}
```

```
    }
    ql.begin();
    ql.pop_front();
    for (list<int>::iterator it=ql.begin();
it!=ql.end(); ++it){
        cout<<*it<<endl;

    }
    return 0;
}
```

```
//Print out all items on the screen
void printall()
{

    ListElem *i;
    i=pfirst;
    if(countitem(>0){
        while(i!=NULL){
            cout<<i->data<<endl;
            i=i->next;
        }
    }
    else cout<<"This is no item.\n";

}

//count the number of items in the stack
int countitem()
{
    ListElem *i;
    int t=0;
    i=pfirst;
    while(i!=NULL){
        t=t+1;
        i=i->next;
    }

    return t;

}

ListElem *find(int tar){
    ListElem *t=pfirst;
    int f = 0;
    while (t != NULL)
    {
        if (t->data==tar) { f = 1; break; }
```

```
        t = t->next;
    }

    if (f != 0) return t;
    else return NULL;
}

ListElem *findmin(){
    ListElem *t=pfirst;
    ListElem *min=pfirst;
    while (t != NULL)
    {
        if (t->data<min->data) { min=t;}
        t = t->next;
    }

    return min;
}

ListElem *findmax(){
    ListElem *t=pfirst;
    ListElem *max=pfirst;
    while (t != NULL)
    {
        if (t->data>max->data) { max=t;}
        t = t->next;
    }

    return max;
}

//Remove an item
ListElem *Pop(){
```

```
if(countitem()>0){ //make sure the list is not empty.
    ListElem *temp,*del;

    if(countitem()==1){ //The list contains only one item
        del=pfirst;
        pfirst=NULL;
        plast=NULL;

    }
    else{ //The list contains more than one item
        temp=pfirst;
        pfirst=pfirst->next;
        del=temp;
        temp=NULL;
    }
    return del;
}else return NULL;

}

void showmenu(){

    cout<<"=====\n";
    cout<<"Stack Operations Menu\n";

    cout<<"=====\n";
    cout<<"1.Add a new item\n";
    cout<<"2.Delete an item\n";
    cout<<"3.Show number of items\n";
    cout<<"4.Show min and max items\n";
    cout<<"5.Find an item\n";
    cout<<"6.Show all items\n";
    cout<<"7.Exit\n";

}

void select(){
```

```
int val, ch;
char yes='y';
ListElem *del;
while(yes=='y'){
    cout<<"Enter your choice:";cin>>ch;
    switch(ch){
        case 1:
            cout<<"Value:";cin>>val;
            Push(val);
            break;
        case 2:
            del=Pop();
            if(del!=NULL) cout<<"Deleted:"<<del->data<<endl;
            break;
        case 3:
            cout<<"Number of items:"<<countitem()<<endl;
            break;
        case 4:
            if(findmin()!=NULL && findmax()!=NULL){
                cout<<"Min item:"<<findmin()->data<<endl;
                cout<<"Max item:"<<findmax()->data<<endl;}
            break;
        case 5:
            cout<<"Find what?";cin>>val;
            if(find(val)!=NULL) cout<<"Found " <<find(val)-
>data<<endl;
            else cout<<"Not found\n";
            break;
        case 6:
            cout<<"All items:\n";
            printall();
            break;
        case 7: break;

        default: cout<<"Invalid choice\n";

    }
    cout<<"Continue?y/n:";cin>>yes;
}
```

<pre>} int main(){ showmenu(); select(); getch(); return 0; }</pre>	
---	--

