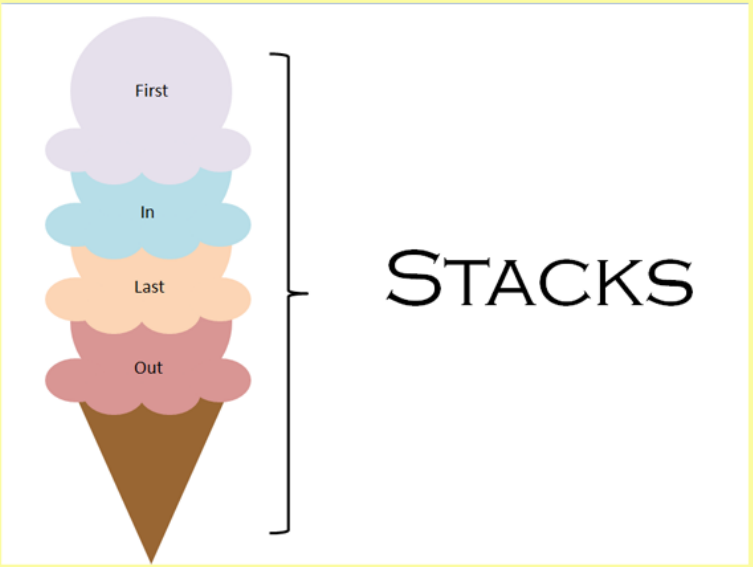
Data Structure

In C Language

Free Online Course

Eng. Mohamed Yousef



Follow me

youtube.com/MohamedYousef2
electronics010.blogspot.com
facebook.com/electronics010

Contents

A

Data Structure in C

Eng. Mohamed Yousef

(Slides Book)

Egypt, 2020

The link of data structure in C course on my youtube channel is:

https://www.youtube.com/playlist?list=PLfgCIULRQavxpi-GYpkLt8_sFb2VlT6Zo

Follow me on:

https://www.youtube.com/mohamedyousef2

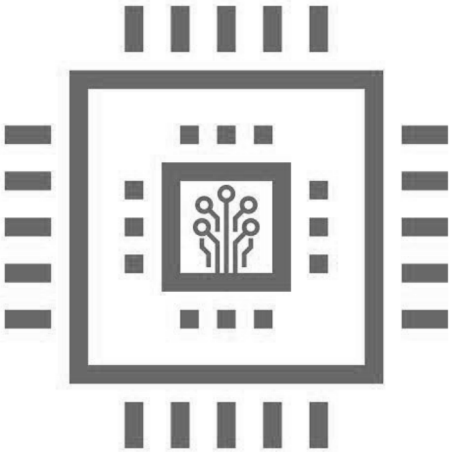
https://electronics010.blogspot.com/

https://www.facebook.com/electronics010

Contents:

Embedded Systems

Basic Data Structures



01- Introduction

Eng. Mohamed Yousef

What is data structure?

Data Structure is the way of storing and access data from memory so that data can be used efficiently.

Actually in our programming data stored in main memory(RAM) and To develop efficient software or firmware we need to care about memory.
To efficiently manage we required data structure.

Types of Data Structure

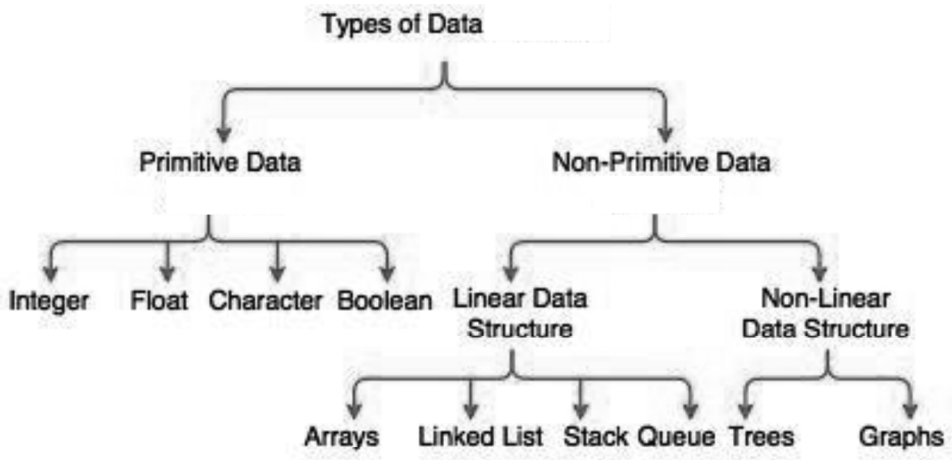


Fig. Types of Data Structure

electronics010.blogspot.com Basic Data Structures Eng. Mohamed Yousef

Types of Data Structure

A. Primitive Data Type

- These data types are used to represent single value.
- It is a basic data type available in most of the programming language.

B. Non-Primitive Data Type

- Data type derived from primary data types are known as Non-Primitive data types.
- Non-Primitive data types are used to store group of values.

It can be divided into two types:

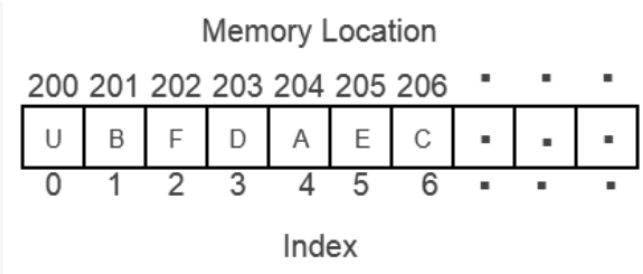
1. Linear Data Structure
2. Non-Linear Data Structure

Types of Data Structure

Linear data structure:

A data structure is said to be linear if items are arranged in linear or sequential format.

Array Data Structure



Linear data structure:

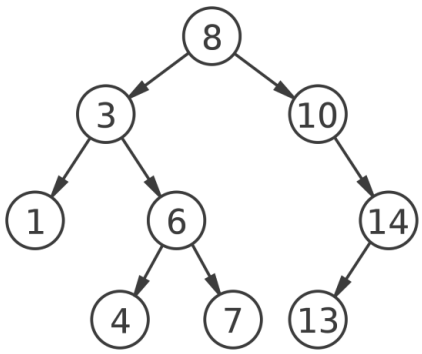
- Array
- Stack
- Queue
- Linked List

Types of Data Structure

Non-linear data structure:

Non-linear data structure does not support sequential format.

Tree data structure



Non-linear data structure:

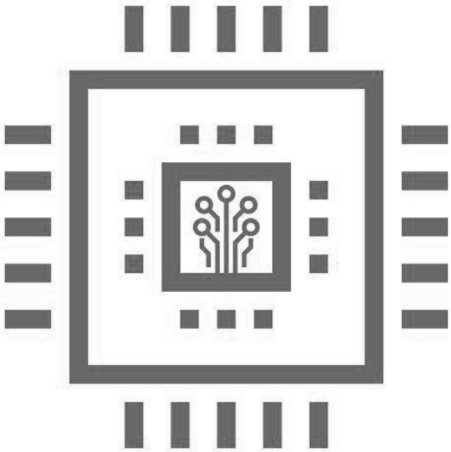
- Tree
- Graph

Following terms are the foundation terms of a data structure.

- **Interface** – Each data structure has an interface. Interface represents the set of operations that a data structure supports. An interface only provides the list of supported operations, type of parameters they can accept and return type of these operations.
- **Implementation** – Implementation provides the internal representation of a data structure.

Embedded Systems

Basic Data Structures

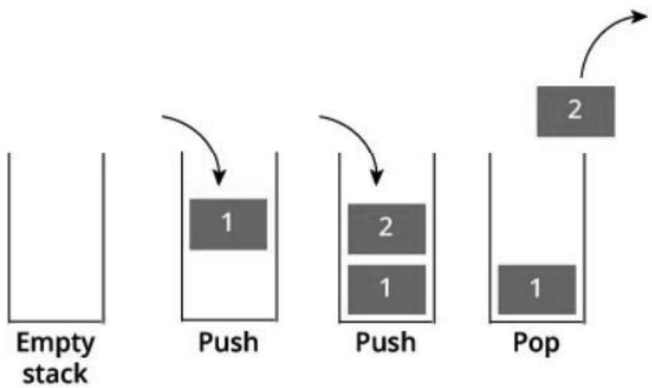


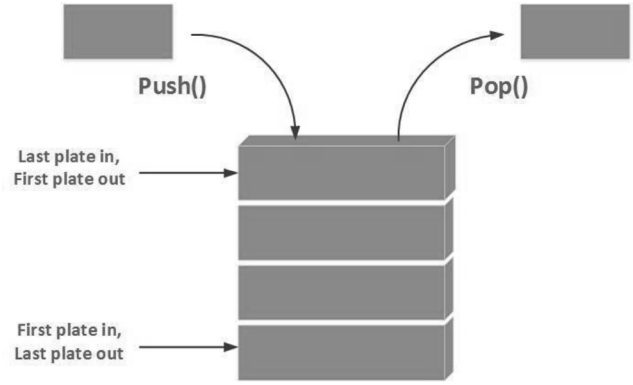
02- Array-based Stack implementation

Eng. Mohamed Yousef

Stack

- Think of a stack as analogous to a pile of dishes.
- When a dish is placed on the pile, it's normally placed at the **top** (referred to as ***pushing*** the dish onto the stack).
- Similarly, when a dish is removed from the pile, it's normally removed from the top (referred to as ***popping*** the dish off the stack).
- Stacks are known as last-in, first-out (***LIFO***) data structures—the last item pushed (inserted) on the stack is the first item popped (removed) from the stack.





What is Stack?

- It is type of **linear data structure**.
- It follows **LIFO** (Last In First Out) property.
- It has only one pointer **TOP** that points the last or top most element of Stack.
- Insertion and Deletion in stack can only be done from top only.
- Insertion in stack is also known as a **PUSH operation**.
- Deletion from stack is also known as **POP operation** in stack.

Definition

"Stack is a collection of similar data items in which both insertion and deletion operations are performed based on LIFO principle".

electronics010.blogspot.com Basic Data Structures Eng. Mohamed Yousef

Stack Implementation

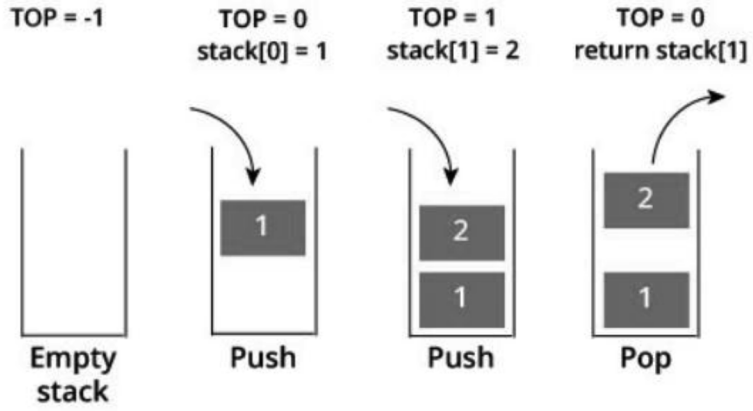
- Stack implementation using array.
- Stack implementation using linked list.

Stack using array.

How stack works

The operations work as follows:

1) A pointer called TOP is used to keep track of the top element in the stack.
2) When initializing the stack, we set its value to -1 so that we can check if the stack is empty by comparing TOP == -1.
3) On pushing an element, we increase the value of TOP and place the new element in the position pointed to by TOP.
4) On popping an element, we return the element pointed to by TOP and reduce its value.
5) Before pushing, we check if stack is already full
6) Before popping, we check if stack is already empty



Stack using Array

- Stack can be implemented using one-dimensional array.
- One-dimensional array is used to hold elements of a stack.
- Implementing a stack using array can store fixed number of data values.
- In a stack, initially top is set to -1.
- Top is used to keep track of the index of the top most element.

Following table shows the Position of Top which indicates the status of stack:

Position of Top	Status of Stack
-1	Stack is empty. (Underflow)
0	Only one element in a stack.
N - 1	Stack is full.
N	Stack is overflow. (Overflow state)

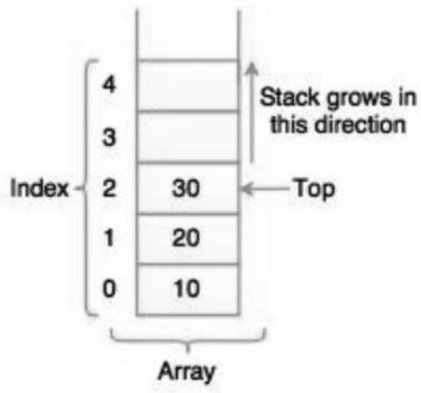


Fig. Implementation Stack using Array

Stack using Array

Stack can be defined using array as follows:

```
typedef struct stack
    {
        int element [MAX];
        int top;
    }stack;
```

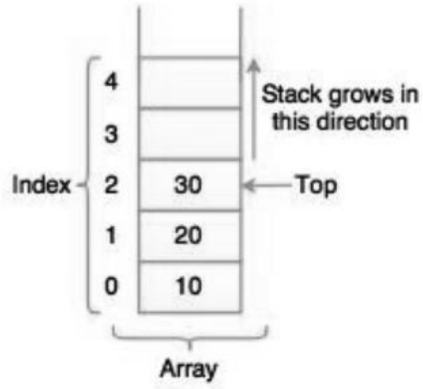


Fig. Implementation Stack using Array

electronics010.blogspot.com Basic Data Structures Eng. Mohamed Yousef

Stack Specification

A stack is data structure that allows the following operations:

- `Push`: Add element to top of stack
- `Pop`: Remove element from top of stack
- `IsEmpty`: Check if stack is empty
- `IsFull`: Check if stack is full

Use of stack

The most common uses of a stack are:

- To reverse a word

Put all the letters in a stack and pop them out.

Because of LIFO order of stack, you will get the letters in reverse order.

- In compilers

Compilers use stack to calculate the value of expressions like 2+4/5*(7-9).

- In browsers

The back button in a browser saves all the urls you have visited previously in a stack. Each time you visit a new page, it is added on top of the stack. When you press the back button, the current URL is removed from the stack and the previous url is accessed.

Stack declaration.

```
#define MAX_SIZE 10
#define DATA_TYPE char

typedef struct{
    DATA_TYPE elements[MAX_SIZE];
    int top;
}Stack;
```

Stack function prototypes.

```
void Init(Stack *);
int IsFull(Stack *);
int IsEmpty(Stack *);
int Push(DATA_TYPE, Stack *);
int Pop(DATA_TYPE *, Stack *);
int StackSize(Stack *);
void ClearStack(Stack *);
```


Stack Implementation.

Initialization of stack.

TOP points to the top-most element of stack.

1) TOP: = -1;

2) Exit

```
void Init(Stack *ptr_stack){

    ptr_stack->top = -1;
}
```

Stack Implementation.

Stack is full.

1) IF TOP = MAX - 1 then
   return:=1;
1) Otherwise
   return:=0;
1) End of IF
2) Exit

```
int IsFull(Stack *ptr_stack){

    if(ptr_stack->top == MAX_SIZE - 1){
        return 1;
    } else {
        return 0;
    }
}
```

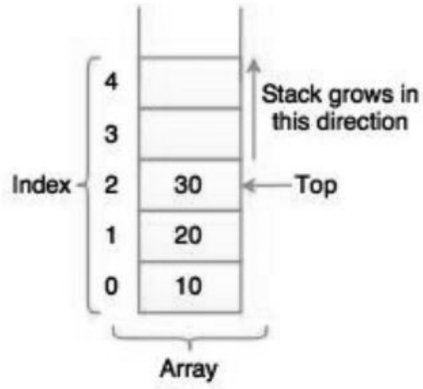


Fig. Implementation Stack using Array

Stack Implementation.

Stack is empty.

1) IF TOP = - 1 then
   return:=1;
2) Otherwise
   return:=0;
3) End of IF
4) Exit

```
int IsEmpty(Stack *ptr_stack){

    if(ptr_stack->top == -1){
        return 1;
    } else {
        return 0;
    }
}
```

Stack Implementation.

Push an item into stack.

1) IF TOP = MAX - 1 then return 0;
   Exit;
2) Otherwise
   TOP: = TOP + 1;
   STACK (TOP):= ITEM;
3) End of IF
1) Exit

```
int Push(DATA_TYPE element, Stack *ptr_stack){

    if(ptr_stack->top == MAX_SIZE - 1){ // Is stack full?
        return 0;
    } else {
        ptr_stack->top++;
        ptr_stack->elements[ptr_stack->top] = element;
        return 1;
    }
}
```

electronics010.blogspot.com Basic Data Structures Eng. Mohamed Yousef

Stack Implementation.

Pop an element from stack.

1) IF TOP = - 1 then return 0;
   Exit;
2) Otherwise
   ITEM: =STACK (TOP);
   TOP: = TOP - 1;
3) End of IF
1) Exit

```
int Pop(DATA_TYPE *ptr_element, Stack *ptr_stack){

    if(ptr_stack->top == -1){ // Is stack empty?
        return 0;
    } else {
        *ptr_element = ptr_stack->elements[ptr_stack->top];
        ptr_stack->top--;
        return 1;
    }
}
```

Stack Implementation.

```
int StackSize(Stack *ptr_stack){

    if(ptr_stack->top == -1){ // Is stack empty?
        return 0;
    } else {
        return (ptr_stack->top + 1);
    }
}
```

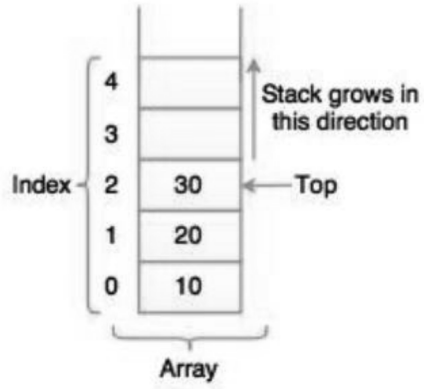


Fig. Implementation Stack using Array

Stack Implementation.

```
void ClearStack(Stack *ptr_stack){
    ptr_stack->top = -1;
}
```

```c
/*
============================================================
==================
PROGRAM     : Array-based stack implementation
Author      : Eng. Mohamed Sayed Yousef
              http://electronics010.blogspot.com.eg/
Date        : November 2018
Version     : 1.0
Description :
============================================================
==================
*/
#include <stdio.h>
#include "lib/stack.h"

//
============================================================
================
int main(){
Stack stack;
Stack *ptr_stack = &stack;

DATA_TYPE element;
DATA_TYPE *ptr_element = &element;

int x = 0, temp = 0;

    Init(ptr_stack);

    // ------------------------------------------------

    Push('S', ptr_stack);
    Push('y', ptr_stack);
    Push('s', ptr_stack);
    Push('t', ptr_stack);
    Push('e', ptr_stack);
    Push('m', ptr_stack);

    // ------------------------------------------------

    if(IsFull(ptr_stack)){

        printf("\n Stack is full.");

    } else if(IsEmpty(ptr_stack)){

        printf("\n Stack is empty.");

    } else {

        printf("\n Stack size: %d", StackSize(ptr_stack));
    }

    printf("\n ---------------");

```

```
52        // ------------------------------------------------
53
54        temp = ptr_stack->top;
55
56        for(x = 0; x <= temp; ++x){
57            Pop(ptr_element, ptr_stack);
58            printf("\n Element: %c", *ptr_element);
59            printf("\n Size: %d", StackSize(ptr_stack));
60            printf("\n ---------------");
61        }
62
63        // ------------------------------------------------
64
65        return 0;
66    }
67    //
      ================================================================
      ================
68
69
```

```
1   /*
2   ================================================================
    ==================
3   PROGRAM     : Array-based stack implementation
4   Author      : Eng. Mohamed Sayed Yousef
5                 http://electronics010.blogspot.com.eg/
6   Date        : November 2018
7   Version     : 1.0
8   Description :
9   ================================================================
    ==================
10  */
11  #ifndef STACK_H_INCLUDED
12  #define STACK_H_INCLUDED
13  // ------------------------------------------------
14
15  #define MAX_SIZE 10
16  #define DATA_TYPE char
17
18  typedef struct{
19      DATA_TYPE elements[MAX_SIZE];
20      int top;
21  }Stack;
22
23  void Init(Stack *);
24  int IsFull(Stack *);
25  int IsEmpty(Stack *);
26  int Push(DATA_TYPE, Stack *);
27  int Pop(DATA_TYPE *, Stack *);
28  int StackSize(Stack *);
29  void ClearStack(Stack *);
30
31  // ------------------------------------------------
32  #endif // STACK_H_INCLUDED
33
```

```
1   /*
2   ============================================================
    ====================
3   PROGRAM     : Array-based stack implementation
4   Author      : Eng. Mohamed Sayed Yousef
5                 http://electronics010.blogspot.com.eg/
6   Date        : November 2018
7   Version     : 1.0
8   Description :
9   ============================================================
    ====================
10  */
11  #include "stack.h"
12
13  void Init(Stack *ptr_stack){
14
15      ptr_stack->top = -1;
16  }
17  //
    ============================================================
    ================
18  int IsFull(Stack *ptr_stack){
19
20      if(ptr_stack->top == MAX_SIZE - 1){
21          return 1;
22      } else {
23          return 0;
24      }
25  }
26  //
    ============================================================
    ================
27  int IsEmpty(Stack *ptr_stack){
28
29      if(ptr_stack->top == -1){
30          return 1;
31      } else {
32          return 0;
33      }
34  }
35  //
    ============================================================
    ================
36  int Push(DATA_TYPE element, Stack *ptr_stack){
37
38      if(ptr_stack->top == MAX_SIZE - 1){ // Is stack full?
39          return 0;
40      } else {
41          ptr_stack->top++;
42          ptr_stack->elements[ptr_stack->top] = element;
43          return 1;
44      }
45  }
46  //
    ============================================================
```

```
    ================
47  int Pop(DATA_TYPE *ptr_element, Stack *ptr_stack){
48
49      if(ptr_stack->top == -1){ // Is stack empty?
50          return 0;
51      } else {
52          *ptr_element = ptr_stack->elements[ptr_stack->top];
53          ptr_stack->top--;
54          return 1;
55      }
56  }
57  //
    ================================================================
    ================
58  int StackSize(Stack *ptr_stack){
59
60      if(ptr_stack->top == -1){ // Is stack empty?
61          return 0;
62      } else {
63          return (ptr_stack->top + 1);
64      }
65  }
66  //
    ================================================================
    ================
67  void ClearStack(Stack *ptr_stack){
68      ptr_stack->top = -1;
69  }
70  //
    ================================================================
    ================
71
72
73
74
```

```
1   /*
2   ============================================================
    ==================
3   PROGRAM     : Array-based stack implementation
4   Author      : Eng. Mohamed Sayed Yousef
5                 http://electronics010.blogspot.com.eg/
6   Date        : November 2018
7   Version     : 1.0
8   Description :
9   ============================================================
    ==================
10  */
11  #include <stdio.h>
12  #include <string.h>
13
14  #include "lib/stack.h"
15
16  //
    ============================================================
    ================
17  int main(){
18  Stack stack;
19  Stack *ptr_stack = &stack;
20
21  char *text = "12345";
22  char inverse[6] = {'\0'};
23
24  int x = 0, temp = 0;
25
26      Init(ptr_stack);
27
28      // ------------------------------------------------
29
30      for(x = 0; x < strlen(text); ++x){
31          Push(*(text + x), ptr_stack);
32      }
33
34      // ------------------------------------------------
35
36      temp = ptr_stack->top;
37
38      for(x = 0; x <= temp; ++x){
39          Pop(&inverse[x], ptr_stack);
40      }
41
42      printf("\n Text: %s", inverse);
43
44      // ------------------------------------------------
45
46      return 0;
47  }
48  //
    ============================================================
    ================
49
```

```
1   /*
2   ================================================================
    ==================
3   PROGRAM     : Array-based stack implementation
4   Author      : Eng. Mohamed Sayed Yousef
5                 http://electronics010.blogspot.com.eg/
6   Date        : November 2018
7   Version     : 1.0
8   Description :
9   ================================================================
    ==================
10  */
11  #ifndef STACK_H_INCLUDED
12  #define STACK_H_INCLUDED
13  // ------------------------------------------------
14
15  #define MAX_SIZE 10
16  #define DATA_TYPE char
17
18  typedef struct{
19      DATA_TYPE elements[MAX_SIZE];
20      int top;
21  }Stack;
22
23  void Init(Stack *);
24  int IsFull(Stack *);
25  int IsEmpty(Stack *);
26  int Push(DATA_TYPE, Stack *);
27  int Pop(DATA_TYPE *, Stack *);
28  int StackSize(Stack *);
29  void ClearStack(Stack *);
30
31  // ------------------------------------------------
32  #endif // STACK_H_INCLUDED
33
```

```
1   /*
2   ============================================================
    ====================
3   PROGRAM     : Array-based stack implementation
4   Author      : Eng. Mohamed Sayed Yousef
5                 http://electronics010.blogspot.com.eg/
6   Date        : November 2018
7   Version     : 1.0
8   Description :
9   ============================================================
    ====================
10  */
11  #include "stack.h"
12
13  void Init(Stack *ptr_stack){
14
15      ptr_stack->top = -1;
16  }
17  //
    ============================================================
    ================
18  int IsFull(Stack *ptr_stack){
19
20      if(ptr_stack->top == MAX_SIZE - 1){
21          return 1;
22      } else {
23          return 0;
24      }
25  }
26  //
    ============================================================
    ================
27  int IsEmpty(Stack *ptr_stack){
28
29      if(ptr_stack->top == -1){
30          return 1;
31      } else {
32          return 0;
33      }
34  }
35  //
    ============================================================
    ================
36  int Push(DATA_TYPE element, Stack *ptr_stack){
37
38      if(ptr_stack->top == MAX_SIZE - 1){ // Is stack full?
39          return 0;
40      } else {
41          ptr_stack->top++;
42          ptr_stack->elements[ptr_stack->top] = element;
43          return 1;
44      }
45  }
46  //
    ============================================================
```

```
    ================
47  int Pop(DATA_TYPE *ptr_element, Stack *ptr_stack){
48
49      if(ptr_stack->top == -1){ // Is stack empty?
50          return 0;
51      } else {
52          *ptr_element = ptr_stack->elements[ptr_stack->top];
53          ptr_stack->top--;
54          return 1;
55      }
56  }
57  //
    ================================================================
    ================
58  int StackSize(Stack *ptr_stack){
59
60      if(ptr_stack->top == -1){ // Is stack empty?
61          return 0;
62      } else {
63          return (ptr_stack->top + 1);
64      }
65  }
66  //
    ================================================================
    ================
67  void ClearStack(Stack *ptr_stack){
68      ptr_stack->top = -1;
69  }
70  //
    ================================================================
    ================
71
72
73
74
```

```
1   /*
2   ================================================================
    ==================
3   PROGRAM     : Array-based stack implementation
4   Author      : Eng. Mohamed Sayed Yousef
5                 http://electronics010.blogspot.com.eg/
6   Date        : November 2018
7   Version     : 1.0
8   Description : Application: Determining Well-Formed
    Expressions with Parenthesis
9   ================================================================
    ==================
10  */
11  #include <stdio.h>
12  #include <string.h>
13
14  #include "lib/stack.h"
15
16  int IsMatchingPair(char opened_char, char closed_char);
17  int IsBalanced(char exp[]);
18  //
    ================================================================
    ================
19  int main(){
20  char text[] = "b-{(x+y)*z+3}";
21
22      if(IsBalanced(text)){
23
24          printf("\n Balanced");
25      } else {
26          printf("\n Not Balanced");
27      }
28
29      return 0;
30  }
31  //
    ================================================================
    ================
32  int IsMatchingPair(char opened_char, char closed_char){
33
34     if (opened_char == '(' && closed_char == ')'){
35          return 1;
36     } else if (opened_char == '{' && closed_char == '}') {
37          return 1;
38     } else if (opened_char == '[' && closed_char == ']') {
39          return 1;
40     } else {
41          return 0;
42     }
43  }
44  //
    ================================================================
    ================
45  int IsBalanced(char exp[]){
46  Stack stack;
```

```
47  Stack *ptr_stack = &stack;
48  int x = 0;
49  char element = '\0';
50  int length = strlen(exp);
51
52      Init(ptr_stack);
53
54      // Test for each character in expression
55      for(x = 0; x < length; ++x){
56
57          //
    ------------------------------------------------------
58          if( exp[x]=='(' || exp[x]=='{' || exp[x]=='['){
59
60              Push(exp[x], ptr_stack);
61
62          } else {
63
64              if( exp[x]==')' || exp[x]=='}' || exp[x]==']'
    ){
65
66                  if(IsEmpty(ptr_stack)){ // there is no
    opening part
67                      return 0;
68                  } else {
69                      Pop(&element, ptr_stack);
70                      if( !IsMatchingPair(element, exp[x]))
    return 0;
71                  }
72              }
73          }
74          //
    ------------------------------------------------------
75      } // END for loop
76
77      // If stack is not empty, then the closing parts is
    less than
78      // the opening parts.
79      if(IsEmpty(ptr_stack)){
80          return 1;
81      } else {
82          return 0;
83      }
84  }
85  //
    ============================================================
    ===============
86
87
```

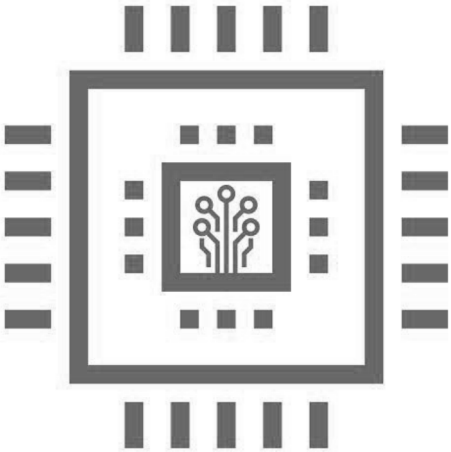
```
1   /*
2   ================================================================
    ==================
3   PROGRAM     : Array-based stack implementation
4   Author      : Eng. Mohamed Sayed Yousef
5                 http://electronics010.blogspot.com.eg/
6   Date        : November 2018
7   Version     : 1.0
8   Description :
9   ================================================================
    ==================
10  */
11  #ifndef STACK_H_INCLUDED
12  #define STACK_H_INCLUDED
13  // ------------------------------------------------
14
15  #define MAX_SIZE 30
16  #define DATA_TYPE char
17
18  typedef struct{
19      DATA_TYPE elements[MAX_SIZE];
20      int top;
21  }Stack;
22
23  void Init(Stack *);
24  int IsFull(Stack *);
25  int IsEmpty(Stack *);
26  int Push(DATA_TYPE, Stack *);
27  int Pop(DATA_TYPE *, Stack *);
28  int StackSize(Stack *);
29  void ClearStack(Stack *);
30
31  // ------------------------------------------------
32  #endif // STACK_H_INCLUDED
33
```

```
1   /*
2   ============================================================
    ====================
3   PROGRAM     : Array-based stack implementation
4   Author      : Eng. Mohamed Sayed Yousef
5                 http://electronics010.blogspot.com.eg/
6   Date        : November 2018
7   Version     : 1.0
8   Description :
9   ============================================================
    ====================
10  */
11  #include "stack.h"
12
13  void Init(Stack *ptr_stack){
14
15      ptr_stack->top = -1;
16  }
17  //
    ============================================================
    ================
18  int IsFull(Stack *ptr_stack){
19
20      if(ptr_stack->top == MAX_SIZE - 1){
21          return 1;
22      } else {
23          return 0;
24      }
25  }
26  //
    ============================================================
    ================
27  int IsEmpty(Stack *ptr_stack){
28
29      if(ptr_stack->top == -1){
30          return 1;
31      } else {
32          return 0;
33      }
34  }
35  //
    ============================================================
    ================
36  int Push(DATA_TYPE element, Stack *ptr_stack){
37
38      if(ptr_stack->top == MAX_SIZE - 1){ // Is stack full?
39          return 0;
40      } else {
41          ptr_stack->top++;
42          ptr_stack->elements[ptr_stack->top] = element;
43          return 1;
44      }
45  }
46  //
    ============================================================
```

```
    ================
47  int Pop(DATA_TYPE *ptr_element, Stack *ptr_stack){
48
49      if(ptr_stack->top == -1){ // Is stack empty?
50          return 0;
51      } else {
52          *ptr_element = ptr_stack->elements[ptr_stack->top];
53          ptr_stack->top--;
54          return 1;
55      }
56  }
57  //
    ================================================================
    ================
58  int StackSize(Stack *ptr_stack){
59
60      if(ptr_stack->top == -1){ // Is stack empty?
61          return 0;
62      } else {
63          return (ptr_stack->top + 1);
64      }
65  }
66  //
    ================================================================
    ================
67  void ClearStack(Stack *ptr_stack){
68      ptr_stack->top = -1;
69  }
70  //
    ================================================================
    ================
71
72
73
74
```

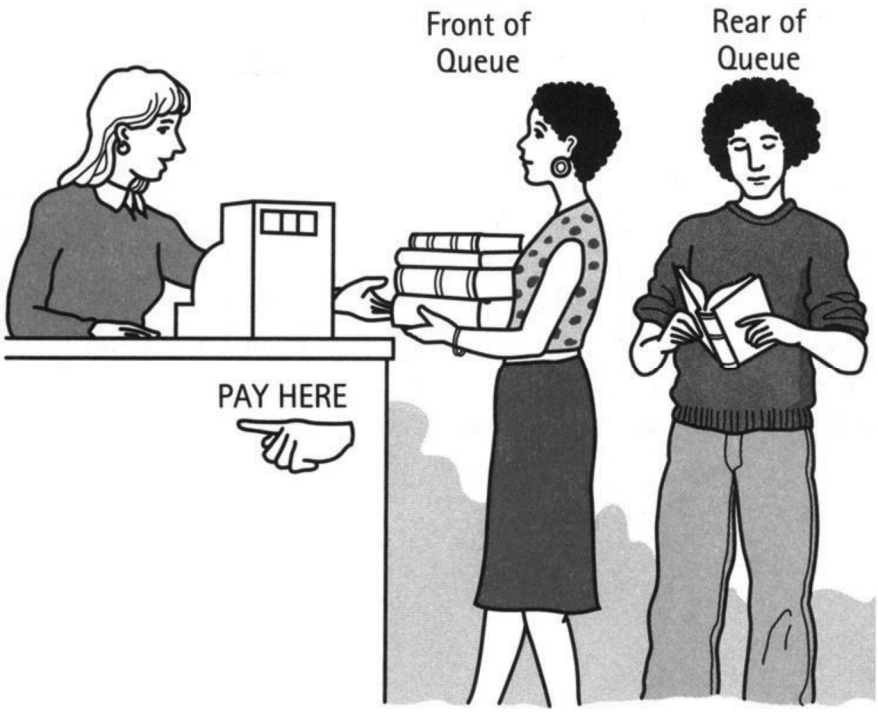
Embedded Systems

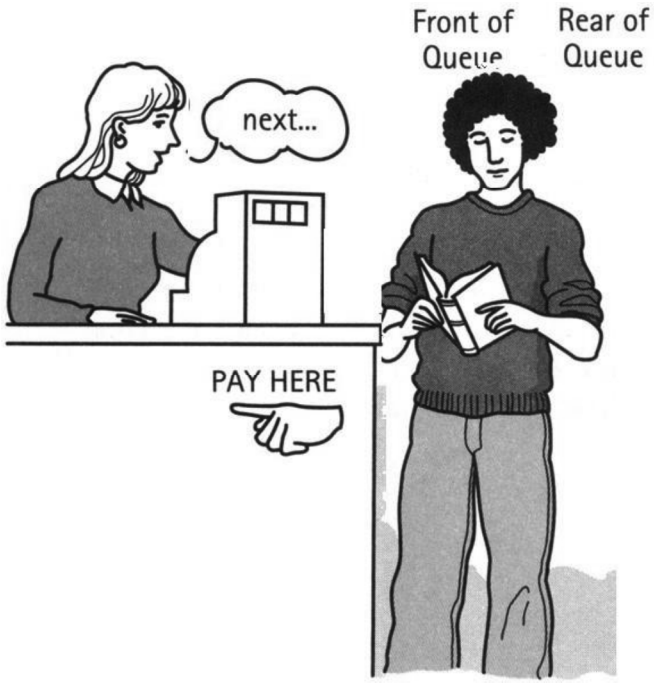
Basic Data Structures

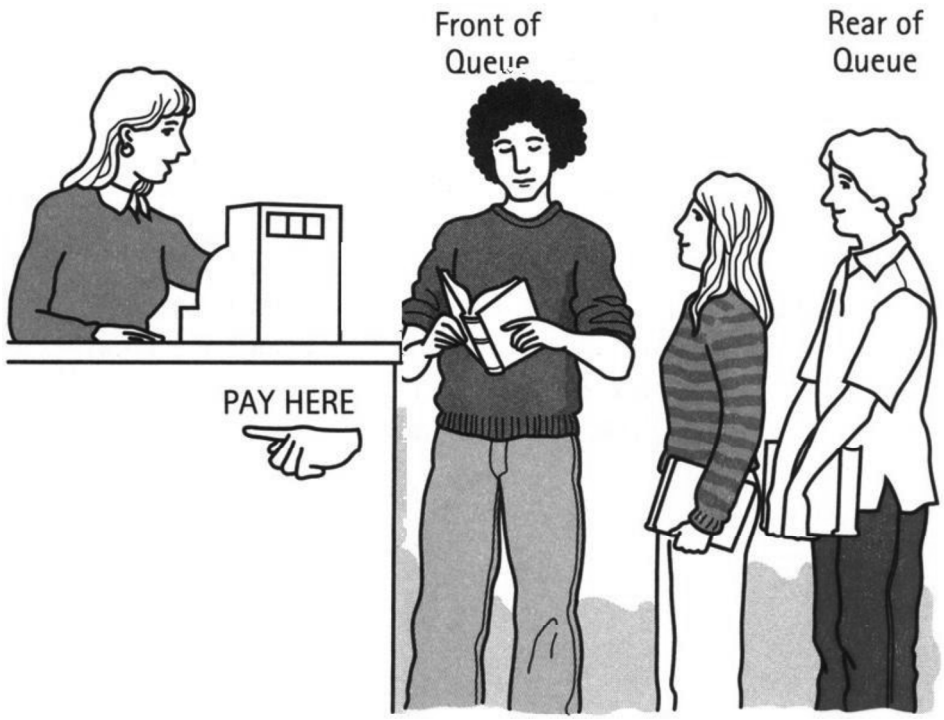


03- Array-based Queue implementation

Eng. Mohamed Yousef


Front of Queue
Rear of Queue
PAY HERE


Front of Queue
Rear of Queue
next...
PAY HERE

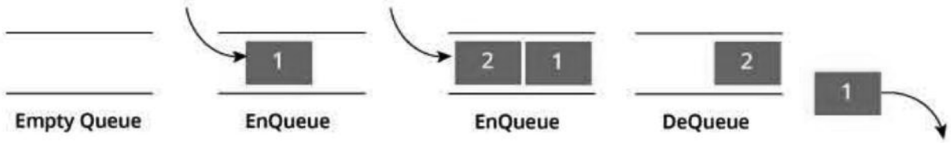


- ✓ Queue
- ✓ Front
- ✓ Rear
- ✓ Enqueue
- ✓ Dequeue
- ✓ First-In-First-Out (FIFO)

- Enqueue: inserts an element into the back of the queue.
- Dequeue: removes an element from the front of the queue.

What is Queue?

- A queue is a collection of data that are added and removed based on the first-in-first-out (FIFO) principle.
- It means that the first element added to the queue will be the first one to be removed from the queue.
- Front points to the beginning of the queue and Rear points to the end of the queue.



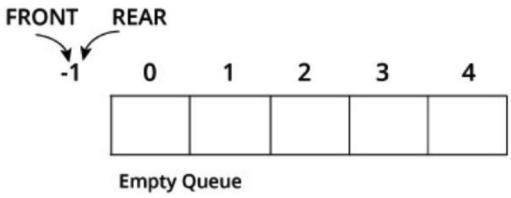
Linear Queue

How Queue Works

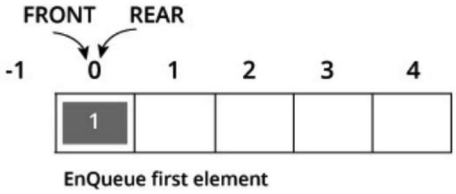
Queue operations work as follows:

1. Two pointers called `FRONT` and `REAR` are used to keep track of the first and last elements in the queue.
2. When initializing the queue, we set the value of `FRONT` and `REAR` to -1.
3. On enqueing an element, we increase the value of `REAR` index and place the new element in the position pointed to by `REAR`.
4. On dequeueing an element, we return the value pointed to by `FRONT` and increase the `FRONT` index.
5. Before enqueing, we check if queue is already full.
6. Before dequeuing, we check if queue is already empty.
7. When enqueing the first element, we set the value of `FRONT` to 0.
8. When dequeing the last element, we reset the values of `FRONT` and `REAR` to -1.

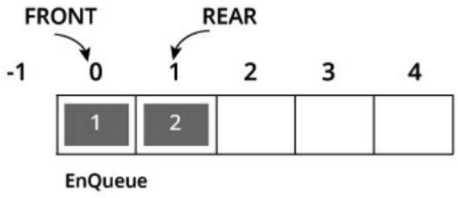
Linear Queue



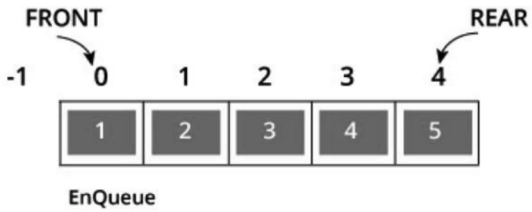
Empty Queue



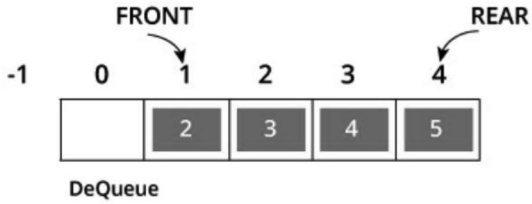
EnQueue first element



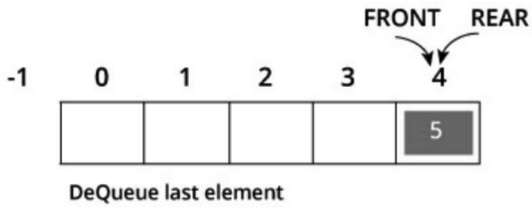
EnQueue



EnQueue



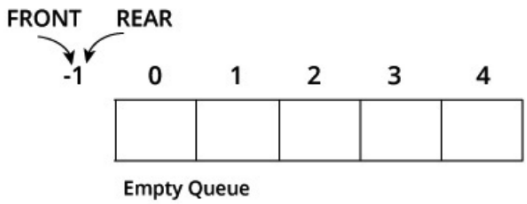
DeQueue



DeQueue last element

electronics010.blogspot.com Basic Data Structures Eng. Mohamed Yousef

Linear Queue

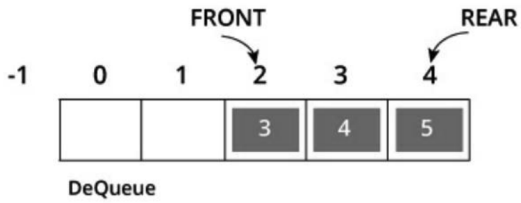


Empty Queue

Linear Queue

Drawback of Linear Queue

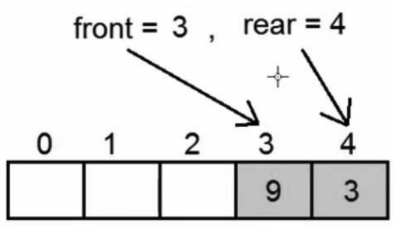
The linear queue suffers from serious drawback that performing some operations, we can not insert items into queue, even if there is space in the queue. Suppose we have queue of 5 elements and we insert 5 items into queue, and then delete some items, then queue has space, but at that condition we can not insert items into queue.

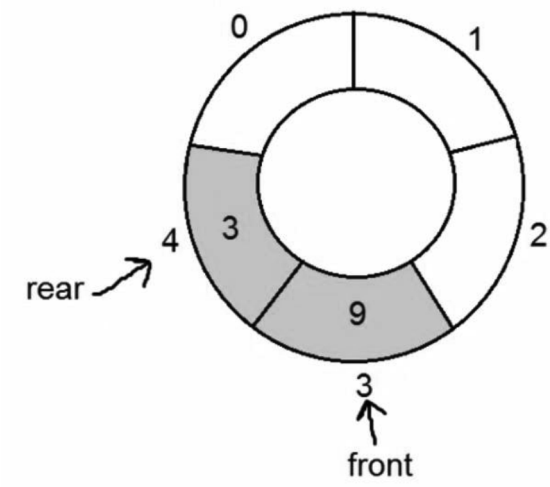


The indexes 0 and 1 can only be used after the queue is reset when all the elements have been dequeued.

Circular Queue

size = 5



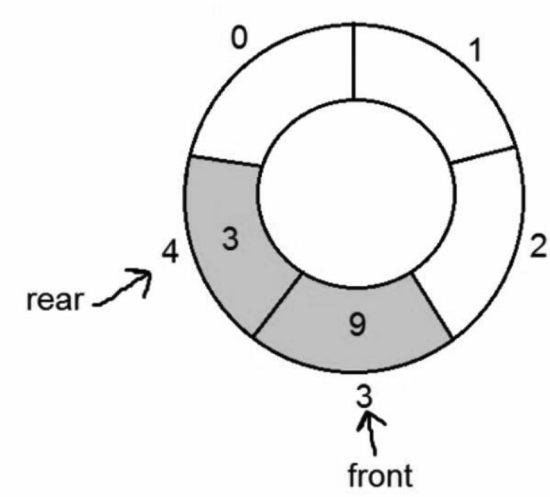


Circular Queue

Enqueue No. → 0/5 = 0 rem (0)
1/5 = 0 rem (1)
2/5 = 0 rem (2)
3/5 = 0 rem (3)
4/5 = 0 rem (4)
5/5 = 1 rem (0)
6/5 = 1 rem (1)
7/5 = 1 rem (2)

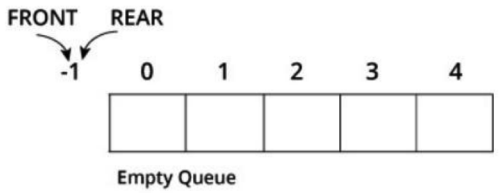
rear index

size = 5

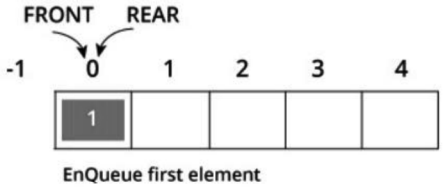



```
rear = (rear+ 1) % size
```

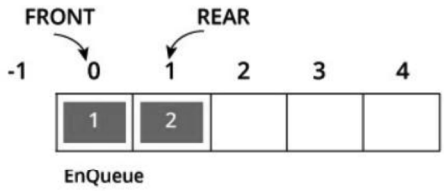

Circular Queue



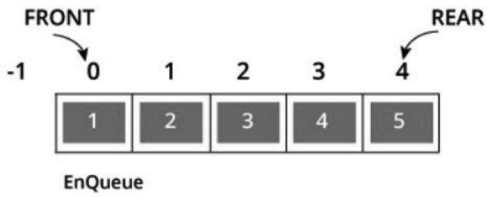
Empty Queue



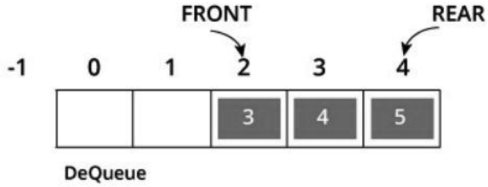
EnQueue first element



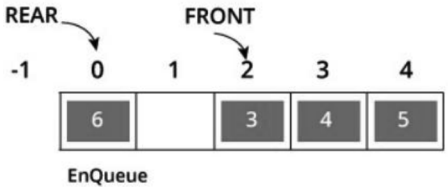
EnQueue



EnQueue

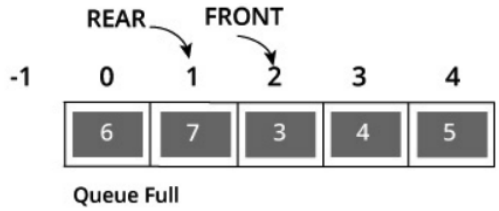


DeQueue



EnQueue

Circular Queue

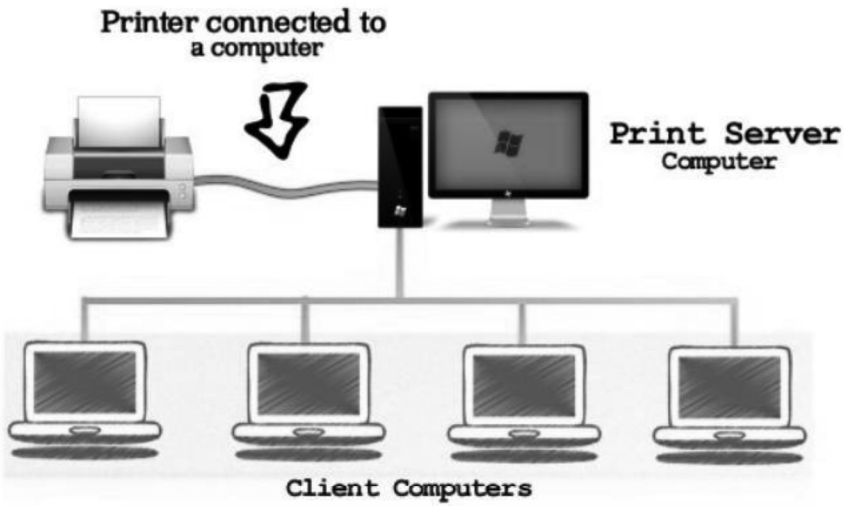



```
IF
     (rear+ 1) % size = front
Then
     queue = full
```


Applications of Queue

Queue is used when things don’t have to be processed immediately, but have to be processed in First In First Out order like .

This property of Queue makes it also useful When a resource is shared among multiple consumers.



Queue declaration.

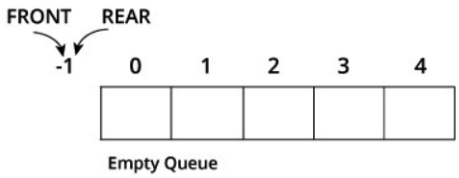
```
#define MAX_SIZE 5
#define DATA_TYPE int

typedef struct{
    DATA_TYPE elements[MAX_SIZE];
    int front;
    int rear;
}Queue;
```


Queue functions prototypes.

```
void Init(Queue *);
int IsFull(Queue *);
int IsEmpty(Queue *);
int Enqueue(DATA_TYPE, Queue *);
int Dequeue(DATA_TYPE *, Queue *);
int QueueSize(Queue *);
void ClearQueue(Queue *);
```

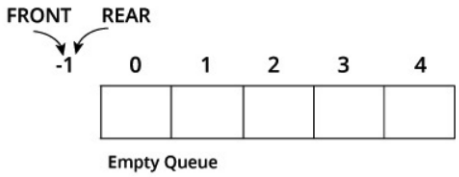
Queue Implementation.

```
void Init(Queue *ptr_queue){

    ptr_queue->front = -1;
    ptr_queue->rear  = -1;
}
```



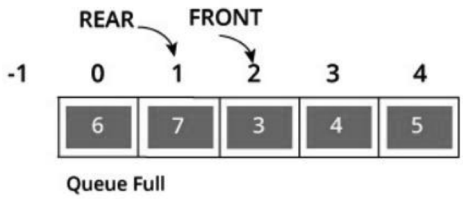
Queue Implementation.

```
int IsEmpty(Queue *ptr_queue){

    if((ptr_queue->front == -1) && (ptr_queue->rear == -1)){
        return 1;
    } else {
        return 0;
    }
}
```



Queue Implementation.

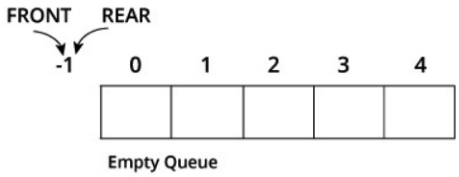
```
int IsFull(Queue *ptr_queue){

    if((ptr_queue->rear + 1) % MAX_SIZE == ptr_queue->front){
        return 1;
    } else {
        return 0;
    }
}
```



Queue Full

Queue Implementation.

```
void ClearQueue(Queue *ptr_queue){

    ptr_queue->front = -1;
    ptr_queue->rear  = -1;
}
```



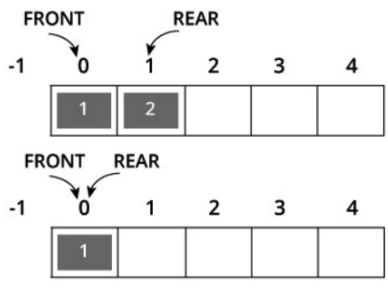
Empty Queue

Queue Implementation.

```
int Enqueue(DATA_TYPE element, Queue *ptr_queue){

    // Is queue full?
    if((ptr_queue->rear + 1) % MAX_SIZE == ptr_queue->front){
        return 0;

    // Is queue empty?
    } else if ((ptr_queue->front == -1) && (ptr_queue->rear == -1)){
        ptr_queue->front = ptr_queue->rear = 0;

    } else {
        ptr_queue->rear = (ptr_queue->rear + 1) % MAX_SIZE;
    }

    ptr_queue->elements[ptr_queue->rear] = element;

    return 1;
}
```

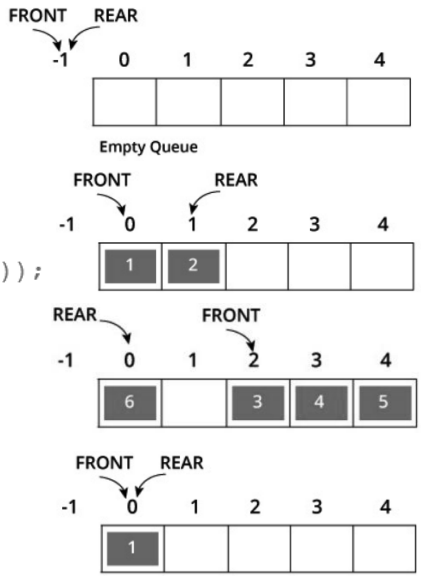

Queue Implementation.

```
int Dequeue(DATA_TYPE *ptr_element, Queue *ptr_queue){

    // Is queue empty?
    if((ptr_queue->front == -1) && (ptr_queue->rear == -1)){

        return 0;
    }

    *ptr_element = ptr_queue->elements[ptr_queue->front];

    // if one element in queue
    if (ptr_queue->front == ptr_queue->rear){

        // reset queue
        ptr_queue->front = ptr_queue->rear = -1;

    } else {
        ptr_queue->front = (ptr_queue->front + 1) % MAX_SIZE;
    }
```



Queue Implementation.

```
int QueueSize(Queue *ptr_queue){

    if((ptr_queue->front == -1) && (ptr_queue->rear == -1)){// Is queue empty?

        return 0;

    } else if(ptr_queue->front < ptr_queue->rear){

        return (ptr_queue->rear - ptr_queue->front +1);

    } else if(ptr_queue->rear < ptr_queue->front){

        return ((MAX_SIZE - ptr_queue->front) + (ptr_queue->rear + 1));

    } else {

        return 1;
    }
```



```c
/*
============================================================
==================
PROGRAM     : Array-based Queue implementation
Author      : Eng. Mohamed Sayed Yousef
              http://electronics010.blogspot.com.eg/
Date        : November 2018
Version     : 1.0
Description :
============================================================
==================
*/
#include <stdio.h>
#include "lib/queue.h"

//
============================================================
================
int main(){
Queue queue;
Queue *ptr_queue = &queue;

DATA_TYPE element;
DATA_TYPE *ptr_element = &element;

int x = 0, temp = 0;

    Init(ptr_queue);

    // ------------------------------------------------

    Enqueue('S', ptr_queue);
    Enqueue('y', ptr_queue);
    Enqueue('s', ptr_queue);
    Enqueue('t', ptr_queue);
    Enqueue('e', ptr_queue);
    Enqueue('m', ptr_queue);

    // ------------------------------------------------

    if(IsFull(ptr_queue)){

        printf("\n Queue is full.");

    } else if(IsEmpty(ptr_queue)){

        printf("\n Queue is empty.");

    } else {

        printf("\n Queue size: %d", QueueSize(ptr_queue));
    }

    printf("\n ---------------");

```

```
52        // ------------------------------------------------------
53
54        temp = ptr_queue->front;
55
56        for(x = temp; x <= ptr_queue->rear; ++x){
57
58            Dequeue(ptr_element, ptr_queue);
59            printf("\n Element: %c", *ptr_element);
60            printf("\n Size: %d", QueueSize(ptr_queue));
61            printf("\n ---------------");
62        }
63
64        // ------------------------------------------------------
65
66        return 0;
67    }
68    //
      ================================================================
      ================
69
70
```

```
1   /*
2   ================================================================
    ==================
3   PROGRAM     : Array-based Queue implementation
4   Author      : Eng. Mohamed Sayed Yousef
5                 http://electronics010.blogspot.com.eg/
6   Date        : November 2018
7   Version     : 1.0
8   Description :
9   ================================================================
    ==================
10  */
11  #ifndef QUEUE_H_INCLUDED
12  #define QUEUE_H_INCLUDED
13  // ------------------------------------------------------
14
15  #define MAX_SIZE 10
16  #define DATA_TYPE char
17
18  typedef struct{
19      DATA_TYPE elements[MAX_SIZE];
20      int front;
21      int rear;
22  }Queue;
23
24  void Init(Queue *);
25  int IsFull(Queue *);
26  int IsEmpty(Queue *);
27  int Enqueue(DATA_TYPE, Queue *);
28  int Dequeue(DATA_TYPE *, Queue *);
29  int QueueSize(Queue *);
30  void ClearQueue(Queue *);
31
32  // ------------------------------------------------------
33  #endif // STACK_H_INCLUDED
34
```

```
1   /*
2   ================================================================
    ==================
3   PROGRAM     : Array-based Queue implementation
4   Author      : Eng. Mohamed Sayed Yousef
5                 http://electronics010.blogspot.com.eg/
6   Date        : November 2018
7   Version     : 1.0
8   Description :
9   ================================================================
    ==================
10  */
11  #include "queue.h"
12
13  void Init(Queue *ptr_queue){
14
15      ptr_queue->front = -1;
16      ptr_queue->rear  = -1;
17  }
18  //
    ================================================================
    ================
19  int IsFull(Queue *ptr_queue){
20
21      if((ptr_queue->rear + 1) % MAX_SIZE ==
    ptr_queue->front){
22          return 1;
23      } else {
24          return 0;
25      }
26  }
27  //
    ================================================================
    ================
28  int IsEmpty(Queue *ptr_queue){
29
30      if((ptr_queue->front == -1) && (ptr_queue->rear ==
    -1)){
31          return 1;
32      } else {
33          return 0;
34      }
35  }
36  //
    ================================================================
    ================
37  int Enqueue(DATA_TYPE element, Queue *ptr_queue){
38
39      // Is queue full?
40      if((ptr_queue->rear + 1) % MAX_SIZE ==
    ptr_queue->front){
41          return 0;
42
43      // Is queue empty?
44      } else if ((ptr_queue->front == -1) &&
```

```
    (ptr_queue->rear == -1)){
45          ptr_queue->front = ptr_queue->rear = 0;
46
47      } else {
48          ptr_queue->rear = (ptr_queue->rear + 1) % MAX_SIZE;
49      }
50
51      ptr_queue->elements[ptr_queue->rear] = element;
52
53      return 1;
54  }
55  //
    ================================================================
    ================
56  int Dequeue(DATA_TYPE *ptr_element, Queue *ptr_queue){
57
58      // Is queue empty?
59      if((ptr_queue->front == -1) && (ptr_queue->rear ==
    -1)){
60
61          return 0;
62      }
63
64      *ptr_element = ptr_queue->elements[ptr_queue->front];
65
66      // if one element in queue
67      if (ptr_queue->front == ptr_queue->rear){
68
69          // reset queue
70          ptr_queue->front = ptr_queue->rear = -1;
71
72      } else {
73          ptr_queue->front = (ptr_queue->front + 1) %
    MAX_SIZE;
74      }
75
76       return 1;
77  }
78  //
    ================================================================
    ================
79  int QueueSize(Queue *ptr_queue){
80
81      if((ptr_queue->front == -1) && (ptr_queue->rear ==
    -1)){// Is queue empty?
82
83          return 0;
84
85      } else if(ptr_queue->front < ptr_queue->rear){
86
87          return (ptr_queue->rear - ptr_queue->front +1);
88
89      } else if(ptr_queue->rear < ptr_queue->front){
90
91          return ((MAX_SIZE - ptr_queue->front) +
```

```
    (ptr_queue->rear + 1));
92
93      } else {
94
95          return 1;
96      }
97  }
98  //
    ================================================================
    ================
99  void ClearQueue(Queue *ptr_queue){
100
101     ptr_queue->front = -1;
102     ptr_queue->rear  = -1;
103 }
104 //
    ================================================================
    ================
105
106
107
108
```

```
1   /*
2   ============================================================
    ====================
3   PROGRAM     : Array-based Queue implementation
4   Author      : Eng. Mohamed Sayed Yousef
5                 http://electronics010.blogspot.com.eg/
6   Date        : November 2018
7   Version     : 1.0
8   Description :
9   ============================================================
    ====================
10  */
11  #include <stdio.h>
12  #include "lib/queue.h"
13
14  void Test_Size(Queue *);
15  //
    ============================================================
    ================
16  int main(){
17  Queue queue;
18  Queue *ptr_queue = &queue;
19
20  DATA_TYPE element;
21  DATA_TYPE *ptr_element = &element;
22
23  int x = 0;
24
25      Init(ptr_queue);
26      Test_Size(ptr_queue);    // Queue is empty
27
28      // ---------------------------------------------------
29
30      for(x=0; x < MAX_SIZE; ++x){
31
32          Enqueue(x, ptr_queue);
33      }
34
35      Test_Size(ptr_queue);    // Queue is full
36
37      // ---------------------------------------------------
38
39      // remove 2 elements
40      Dequeue(ptr_element, ptr_queue);
41      Dequeue(ptr_element, ptr_queue);
42
43      Test_Size(ptr_queue);
44
45      // ---------------------------------------------------
46
47      // add 2 elements
48      Enqueue(11, ptr_queue);
49      Enqueue(12, ptr_queue);
50
51      Test_Size(ptr_queue);    // Queue is full
```

```
52
53      // ------------------------------------------------
54
55      printf("\n\n ===================================");
56      printf("\n Front\tElement\tSize after Dequeue");
57      printf("\n ===================================");
58
59      for(x = 0; x < MAX_SIZE; ++x){
60
61          printf("\n  %d",ptr_queue->front);
62          Dequeue(ptr_element, ptr_queue);
63          printf("\t %d", *ptr_element);
64          printf("\t\t%d", QueueSize(ptr_queue));
65          printf("\n ---------------------------------");
66      }
67
68      // ------------------------------------------------
69
70      return 0;
71  }
72  //
    ================================================================
    ================
73  void Test_Size(Queue *ptr_queue){
74
75      if(IsFull(ptr_queue)){
76
77          printf("\n Queue is full.");
78
79      } else if(IsEmpty(ptr_queue)){
80
81          printf("\n Queue is empty.");
82
83      } else {
84
85          printf("\n Queue size: %d", QueueSize(ptr_queue));
86      }
87
88      printf("\n ---------------");
89  }
90  //
    ================================================================
    ================
91
```

```
1   /*
2   ================================================================
    ==================
3   PROGRAM     : Array-based Queue implementation
4   Author      : Eng. Mohamed Sayed Yousef
5                 http://electronics010.blogspot.com.eg/
6   Date        : November 2018
7   Version     : 1.0
8   Description :
9   ================================================================
    ==================
10  */
11  #ifndef QUEUE_H_INCLUDED
12  #define QUEUE_H_INCLUDED
13  // ------------------------------------------------
14
15  #define MAX_SIZE 5
16  #define DATA_TYPE int
17
18  typedef struct{
19      DATA_TYPE elements[MAX_SIZE];
20      int front;
21      int rear;
22  }Queue;
23
24  void Init(Queue *);
25  int IsFull(Queue *);
26  int IsEmpty(Queue *);
27  int Enqueue(DATA_TYPE, Queue *);
28  int Dequeue(DATA_TYPE *, Queue *);
29  int QueueSize(Queue *);
30  void ClearQueue(Queue *);
31
32  // ------------------------------------------------
33  #endif // STACK_H_INCLUDED
34
```

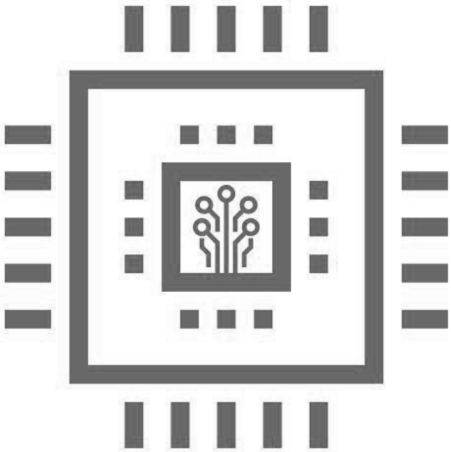
```
1   /*
2   ================================================================
    ==================
3   PROGRAM     : Array-based Queue implementation
4   Author      : Eng. Mohamed Sayed Yousef
5                 http://electronics010.blogspot.com.eg/
6   Date        : November 2018
7   Version     : 1.0
8   Description :
9   ================================================================
    ==================
10  */
11  #include "queue.h"
12
13  void Init(Queue *ptr_queue){
14
15      ptr_queue->front = -1;
16      ptr_queue->rear  = -1;
17  }
18  //
    ================================================================
    ================
19  int IsFull(Queue *ptr_queue){
20
21      if((ptr_queue->rear + 1) % MAX_SIZE ==
    ptr_queue->front){
22          return 1;
23      } else {
24          return 0;
25      }
26  }
27  //
    ================================================================
    ================
28  int IsEmpty(Queue *ptr_queue){
29
30      if((ptr_queue->front == -1) && (ptr_queue->rear ==
    -1)){
31          return 1;
32      } else {
33          return 0;
34      }
35  }
36  //
    ================================================================
    ================
37  int Enqueue(DATA_TYPE element, Queue *ptr_queue){
38
39      // Is queue full?
40      if((ptr_queue->rear + 1) % MAX_SIZE ==
    ptr_queue->front){
41          return 0;
42
43      // Is queue empty?
44      } else if ((ptr_queue->front == -1) &&
```

```
    (ptr_queue->rear == -1)){
45          ptr_queue->front = ptr_queue->rear = 0;
46
47      } else {
48          ptr_queue->rear = (ptr_queue->rear + 1) % MAX_SIZE;
49      }
50
51      ptr_queue->elements[ptr_queue->rear] = element;
52
53      return 1;
54  }
55  //
    ================================================================
    ===============
56  int Dequeue(DATA_TYPE *ptr_element, Queue *ptr_queue){
57
58      // Is queue empty?
59      if((ptr_queue->front == -1) && (ptr_queue->rear ==
    -1)){
60
61          return 0;
62      }
63
64      *ptr_element = ptr_queue->elements[ptr_queue->front];
65
66      // if one element in queue
67      if (ptr_queue->front == ptr_queue->rear){
68
69          // reset queue
70          ptr_queue->front = ptr_queue->rear = -1;
71
72      } else {
73          ptr_queue->front = (ptr_queue->front + 1) %
    MAX_SIZE;
74      }
75
76       return 1;
77
78  }
79  //
    ================================================================
    ===============
80  int QueueSize(Queue *ptr_queue){
81
82      if((ptr_queue->front == -1) && (ptr_queue->rear ==
    -1)){// Is queue empty?
83
84          return 0;
85
86      } else if(ptr_queue->front < ptr_queue->rear){
87
88          return (ptr_queue->rear - ptr_queue->front +1);
89
90      } else if(ptr_queue->rear < ptr_queue->front){
91
```

```
 92            return ((MAX_SIZE - ptr_queue->front) +
     (ptr_queue->rear + 1));
 93
 94        } else {
 95
 96            return 1;
 97        }
 98
 99
100    }
101    //
       ================================================================
       ================
102    void ClearQueue(Queue *ptr_queue){
103
104        ptr_queue->front = -1;
105        ptr_queue->rear  = -1;
106    }
107    //
       ================================================================
       ================
108
109
110
111
```

Embedded Systems

Basic Data Structures

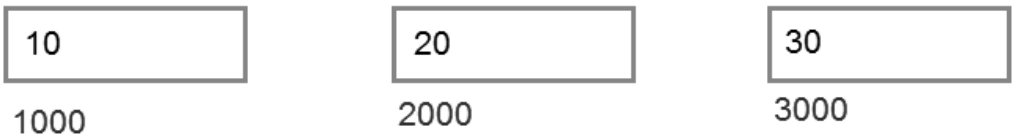


04- Linked List Implementation

Eng. Mohamed Yousef

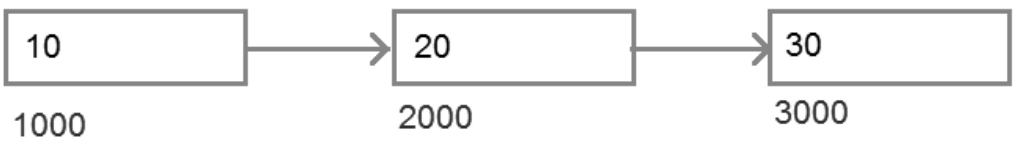
Linked List

Linked List

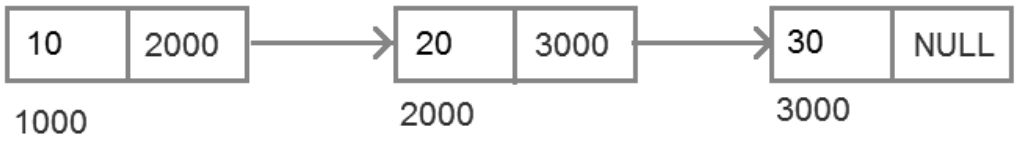


electronics010.blogspot.com Basic Data Structures Eng. Mohamed Yousef

Linked List

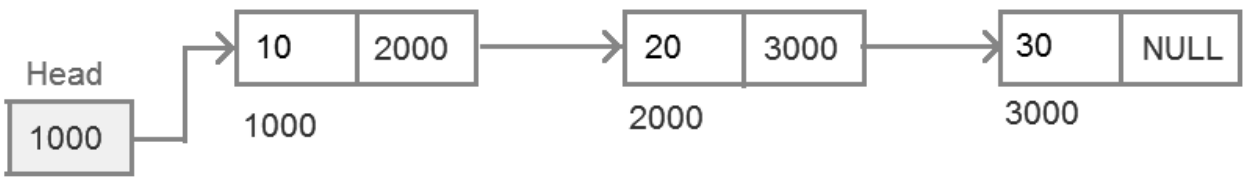


Linked List

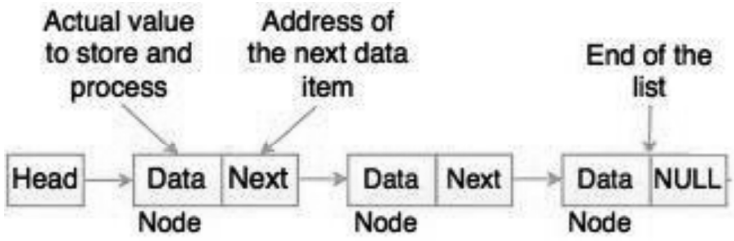


electronics010.blogspot.com Basic Data Structures Eng. Mohamed Yousef

Linked List



Linked List



It is a collection of data elements, called nodes pointing to the next node by means of a pointer.

Linked list contains data items connected together via links.

It can be visualized as a chain of nodes, where every node points to the next node.

Link field is called next and each link is linked with its next link.

Last link carries a link to null to mark the end of the list.

Entry point into the linked list is called the head of the list.

Note: *Head is not a separate node but it is a reference to the first node.*

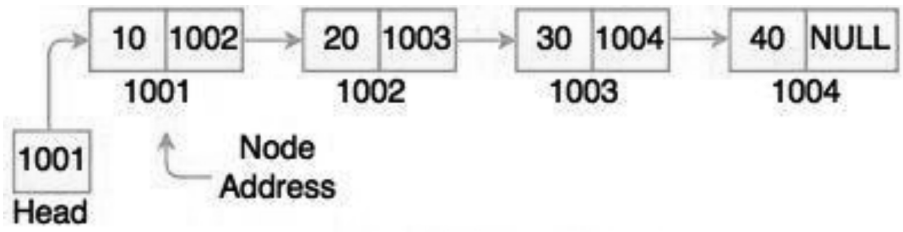
If the list is empty, the head is a null reference.

While accessing a particular item, start at the head and follow the references until you get that data item.

Linked list is used while dealing with an unknown number of objects

electronics010.blogspot.com Basic Data Structures Eng. Mohamed Yousef

Linked List



The address of the first node is always store in a reference node known as Head or Front.

Reference part of the last node must be null.

Linked List

Advantages of Linked List:

- Linked list is dynamic in nature which allocates the memory when required.
- Insert and delete operation can be easily implemented in linked list.

Disadvantages of Linked List:

- Linked list has to access each node sequentially; no element can be accessed randomly.
- In linked list, the memory is wasted as pointer requires extra memory for storage.

Types of Linked List

1. Singly Linked List
2. Doubly Linked List
3. Circular Linked List
4. Doubly Circular Linked List

electronics010.blogspot.com Basic Data Structures Eng. Mohamed Yousef

Linked List declaration

```
#define DATA_TYPE char

typedef struct node{
    DATA_TYPE element;
    struct node *next;
}Node;

Node *head;
```

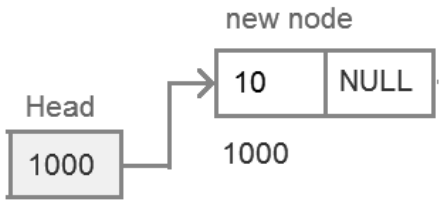
Linked List functions prototypes.

```
Node* Append(Node *, DATA_TYPE);
int Length(Node *);
int Get_Element(Node *, int, DATA_TYPE *);
Node* Delete_First_Node(Node *);
int Delete_Node(Node *, int);
Node* Insert_Frist(Node *, DATA_TYPE);
int Insert_After(Node *, int, DATA_TYPE);
```


Linked List Implementation.

```
Node* Append (Node *head, DATA_TYPE element)
```

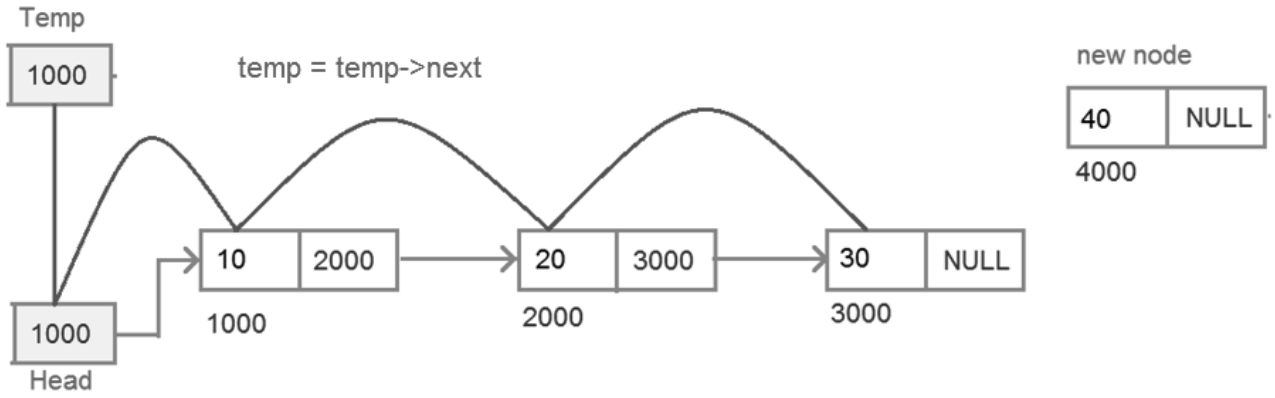
IF linked list is empty



Linked List Implementation.

```
]Node* Append (Node *head, DATA_TYPE element)
```

IF linked list is not empty

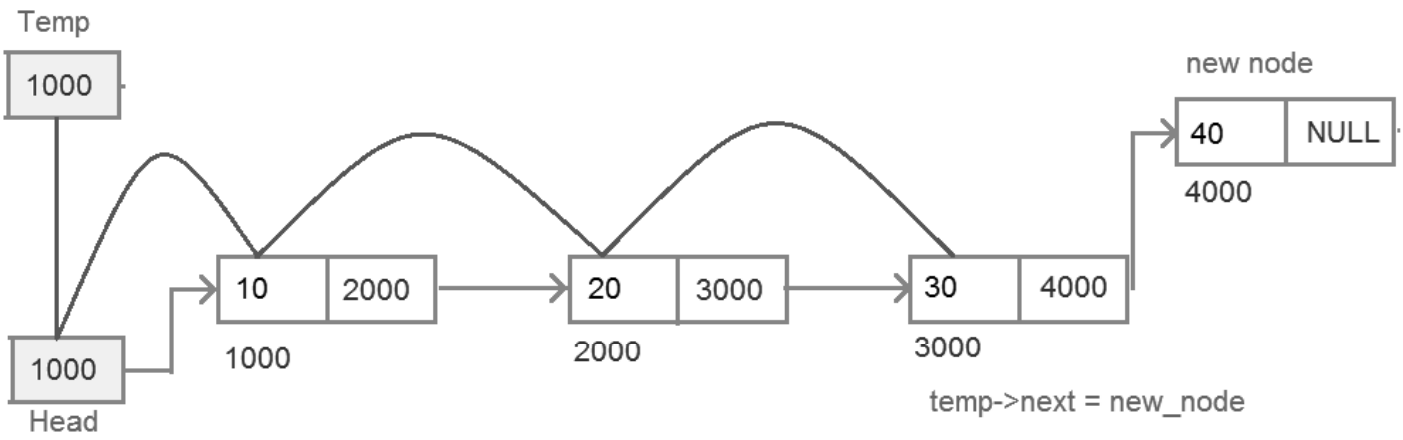


electronics010.blogspot.com Basic Data Structures Eng. Mohamed Yousef

Linked List Implementation.

```
]Node* Append (Node *head, DATA_TYPE element)
```

IF linked list is not empty



Linked List Implementation.

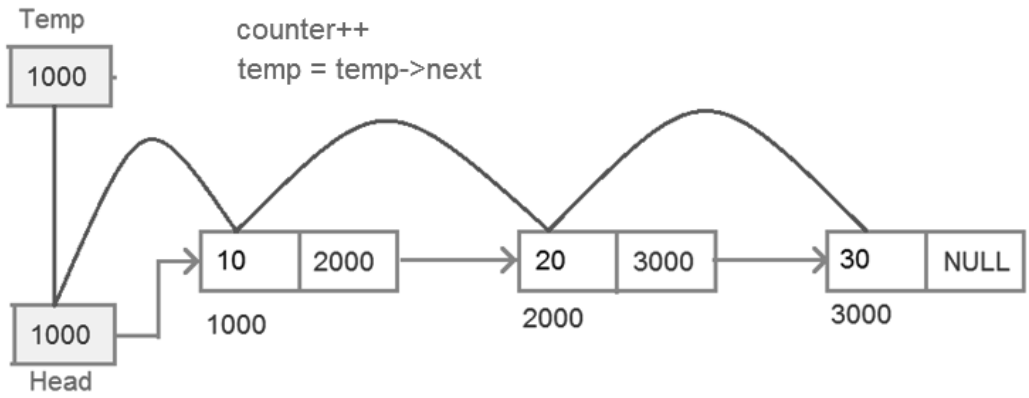
```
Node* Append (Node *head, DATA_TYPE element){
Node *new_node = NULL;    // Local (in stack)

    // create a node
    // reserve a block in heap
    new_node =(Node *)malloc(sizeof(Node));

    if(new_node == NULL){
        return NULL;
    }

    // set data into a new node
    new_node->element = element;
    new_node->next = NULL;
```


Linked List Implementation.

```
    // link the new node to the 'List'
    if(head != NULL){ // linked list is not empty
        // set address in head to 'temp'
        // this address points to 1st node
        Node *temp = head;

        while(temp->next != NULL){
            temp = temp->next;
        }

        temp->next = new_node;
    }

    // return address of new 'Node'
    return new_node;
}
```

Linked List Implementation.

```
int Length(Node *head)
```



Linked List Implementation.

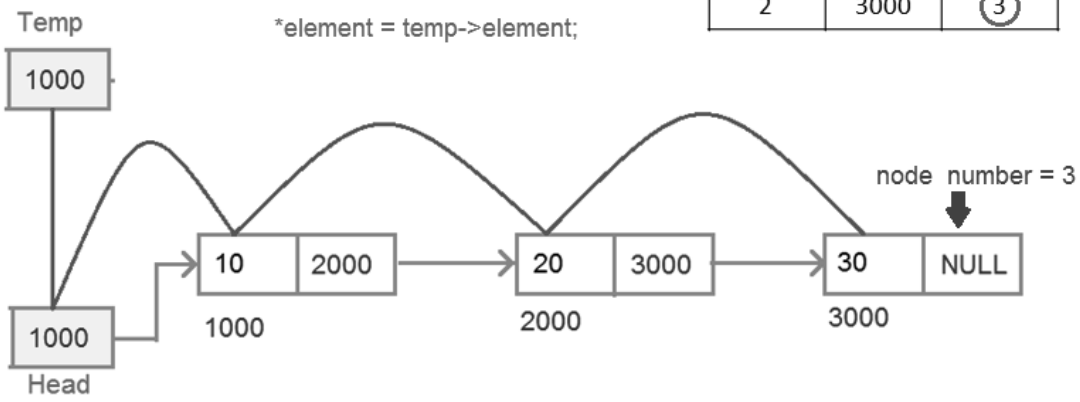
```
int Length(Node *head){
int len = 1;

    if(head == NULL){ // linked list is empty
        return 0;
    }
    // set address in head to 'temp'
    // this address points to 1st node
    Node *temp = head;

    while(temp->next != NULL){

        len++;
        // move temp to the next node
        temp = temp->next;
    }

    return len;
}
```

Linked List Implementation.

```
int Get_Element(Node *head, int node_number, DATA_TYPE *element)
```

```
int x = 1;

while(x < node_number){
    temp = temp->next;
    x++;
}

*element = temp->element;
```

x	address	node
1	2000	2
2	3000	3



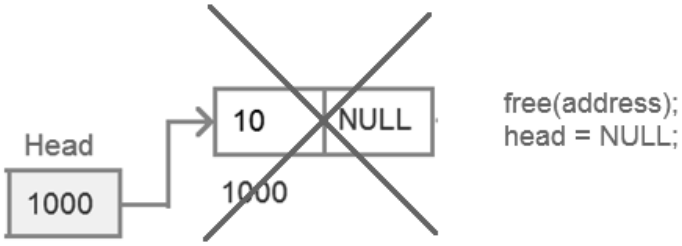
electronics010.blogspot.com Basic Data Structures Eng. Mohamed Yousef

Linked List Implementation.

```
int Get_Element(Node *head, int node_number, DATA_TYPE *element){

    if(head == NULL){ // linked list is empty
        return 0;
    }

    if((node_number > Length(head)) || (node_number < 1)){
        return -1;
    }
    // ------------------------------------------------

    // set address in head to 'temp'
    // this address points to 1st node
    Node *temp = head;
    int x = 1;

    while(x < node_number){
        // move temp to the next node
        temp = temp->next;
        x++;
    }

    *element = temp->element;

    return 1;
}
```

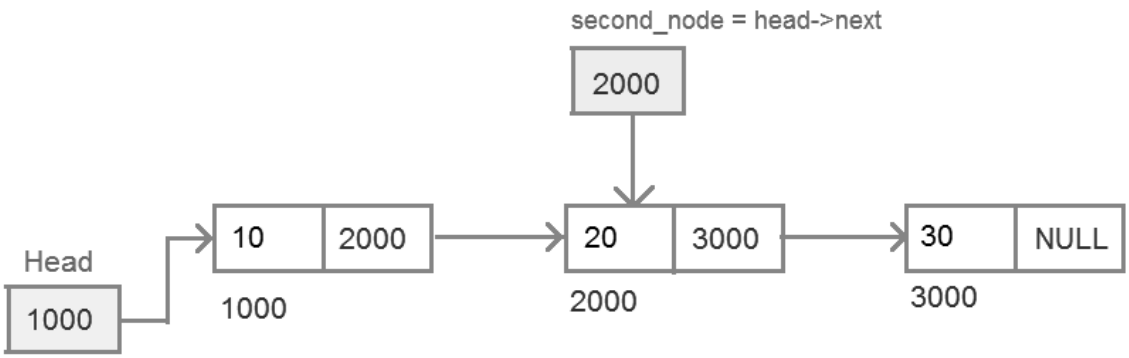
Linked List Implementation.

```
Node* Delete_First_Node(Node *head)
```



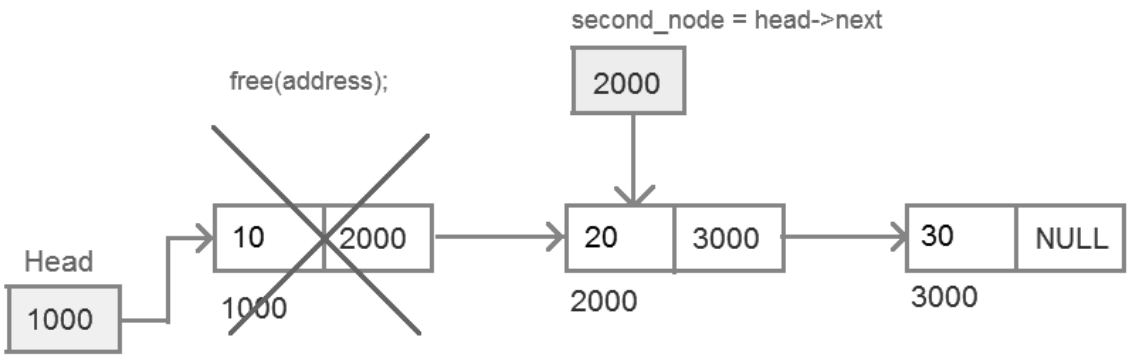
Linked List Implementation.

```
Node* Delete_First_Node(Node *head)
```



Linked List Implementation.

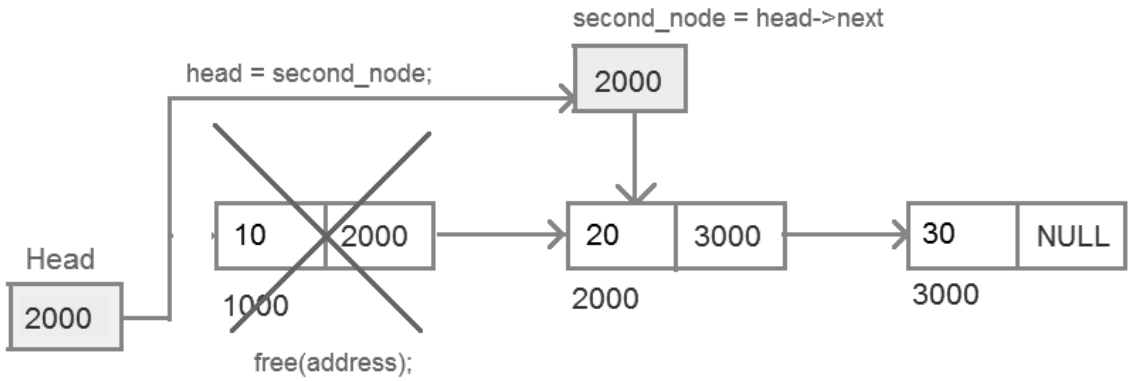
```
Node* Delete_First_Node(Node *head)
```



electronics010.blogspot.com Basic Data Structures Eng. Mohamed Yousef

Linked List Implementation.

```
Node* Delete_First_Node(Node *head)
```



Linked List Implementation.

```
Node* Delete_First_Node(Node *head){

    // linked list is empty
    if(head == NULL){
        return NULL;
    }

    // check List length
    if(Length(head) == 1){
        free(head);
        return NULL;
    } else { // more than one node
        Node *second_node;
        second_node = head->next;
        free(head);
        return second_node;
    }
}
```

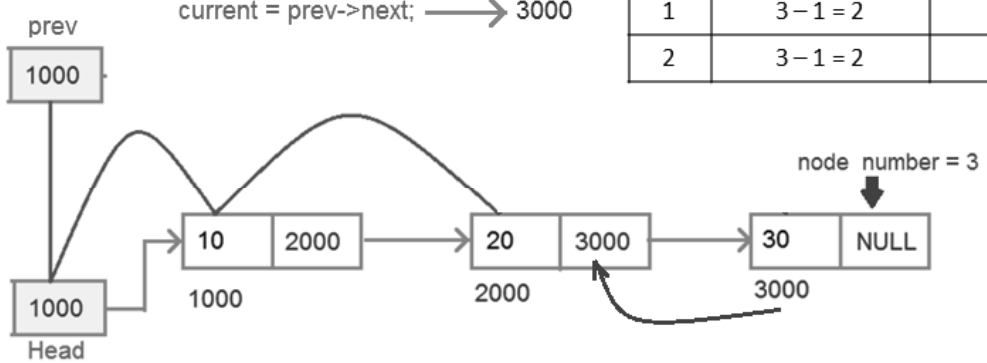

Linked List Implementation.

```
int Delete_Node(Node *head, int node_number)
```

```
Node *prev = head, *current;
   int x = 1;

   while(x < node_number - 1){
      prev = prev->next;
      x++;
   }

   current = prev->next;
```

→ 3000

x	node_number -1	prev = prev_next
1	3 – 1 = 2	2000
2	3 – 1 = 2	



Linked List Implementation.

```
int Delete_Node(Node *head, int node_number)
```

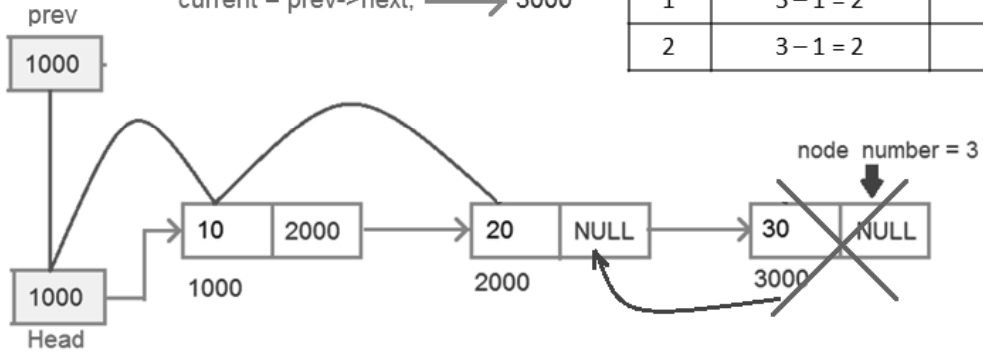
```
Node *prev = head, *current;
   int x = 1;

   while(x < node_number - 1){
      prev = prev->next;
      x++;
   }

   current = prev->next;
```

→ 3000

```
prev->next = current->next;
free(current);
```

→ prev->next = NULL

x	node_number -1	prev = prev_next
1	3 – 1 = 2	2000
2	3 – 1 = 2	



Linked List Implementation.

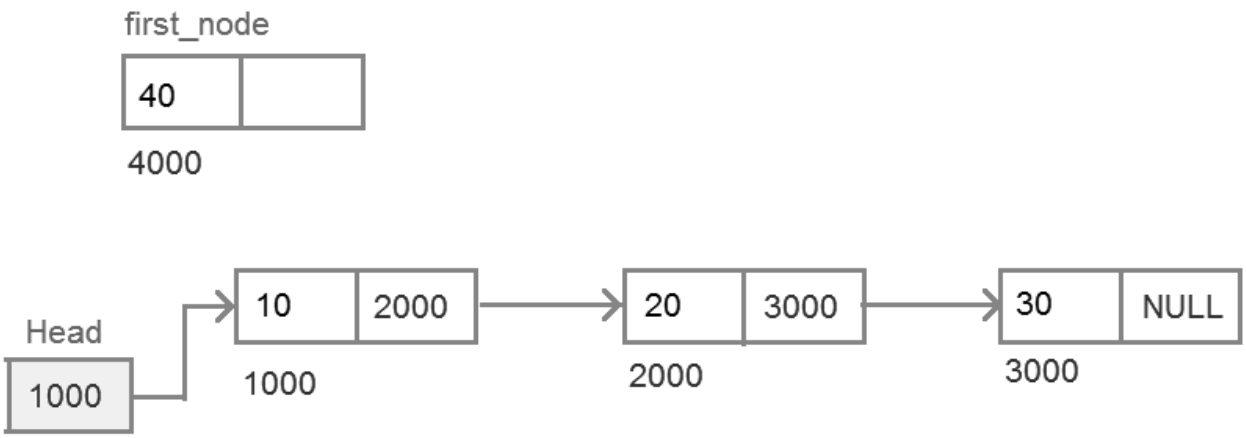
```
int Delete_Node(Node *head, int node_number){

    // linked list is empty
    if((head == NULL)){
        return 0;
    }

    // Invalid node number
    if((node_number > Length(head)) || (node_number < 2)){
        return -1;
    }
    // ------------------------------------------------
```

Linked List Implementation.

```
    // delete any node in the middle or in last
    Node *prev = head, *current;
    int x = 1;

    while(x < node_number - 1){
        prev = prev->next;
        x++;
    }

    current = prev->next;
    prev->next = current->next;
    free(current);

    return 1;
}
```

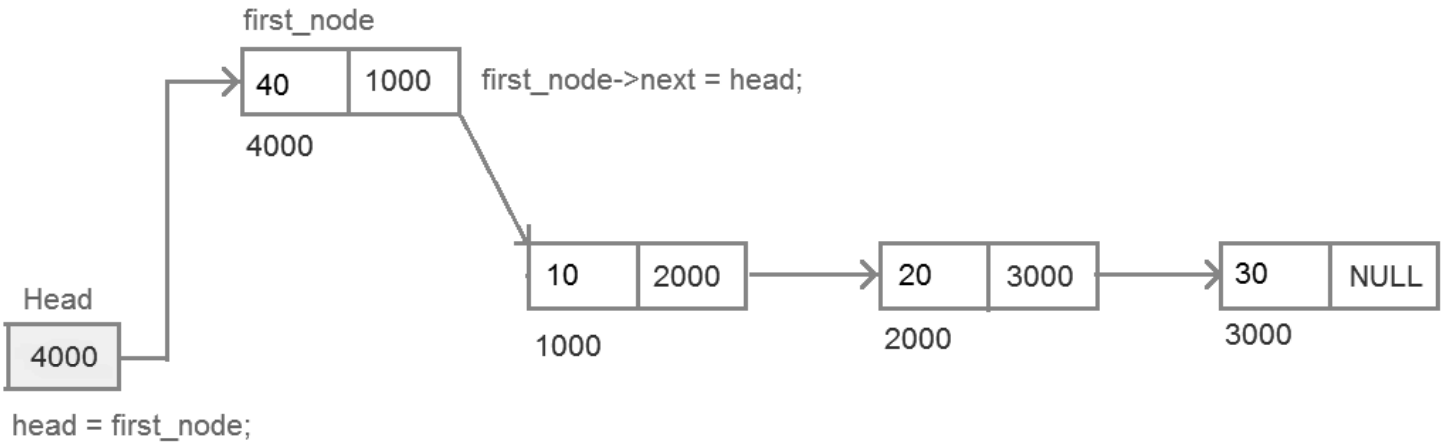

Linked List Implementation.

```
Node* Insert_Frist(Node *head, DATA_TYPE element)
```



Linked List Implementation.

```
Node* Insert_Frist(Node *head, DATA_TYPE element)
```



Linked List Implementation.

```
Node* Insert_Frist(Node *head, DATA_TYPE element){

    // linked list is empty
    if(head == NULL){
        return NULL;
    }

    Node *first_node;

    // create a node
    // reserve a block in heap
    first_node =(Node *)malloc(sizeof(Node));

    if(first_node == NULL){
        return head;
    }
```

Linked List Implementation.

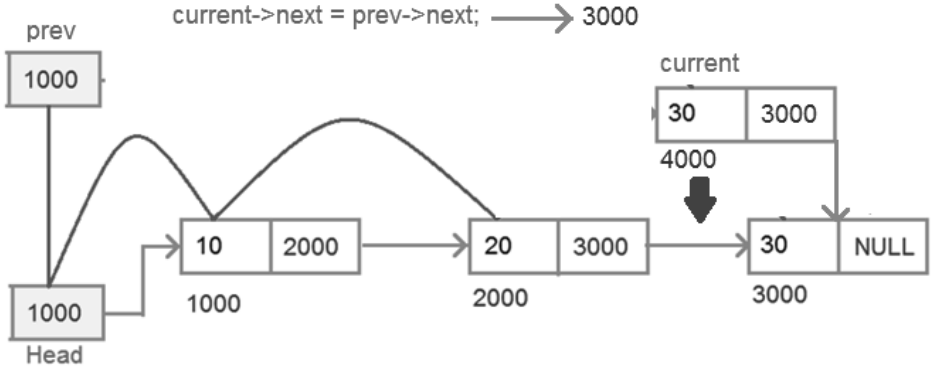
```
    // set data into a new node
    first_node->element = element;
    first_node->next = head;

    head = first_node;

    return head;
}
```


Linked List Implementation.

```
int Insert_After(Node *head, int node_number, DATA_TYPE element)
```

```
Node *prev = head, *current;
  int x = 1;

  while(x < node_number ){
     prev = prev->next;
     x++;
  }

  current->next = prev->next;
```

x	node_number	prev = prev_next
1	2	2000
2	2	



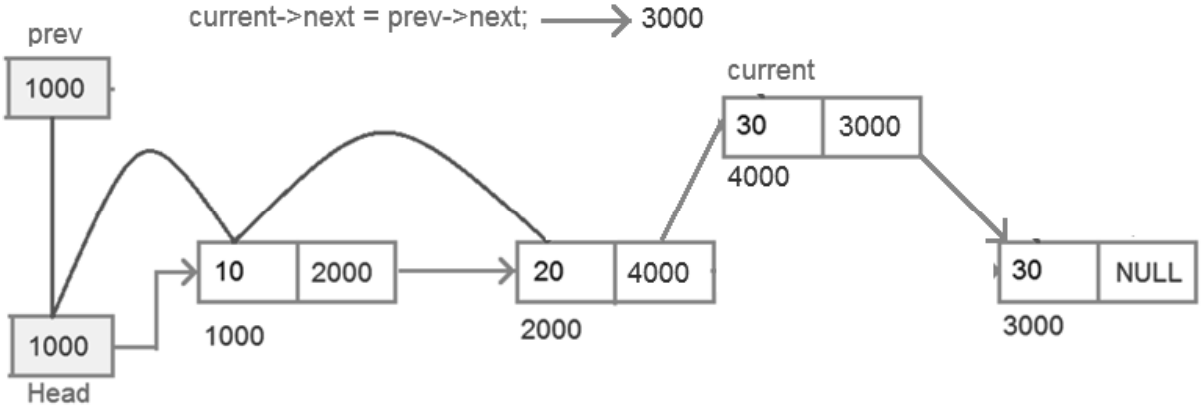
Linked List Implementation.

```
Node *prev = head, *current;
   int x = 1;

   while(x < node_number ){
      prev = prev->next;
      x++;
   }
```

prev->next = current; ⟶ 4000

x	node_number	prev = prev_next
1	2	2000
2	2	

current->next = prev->next; ⟶ 3000



Linked List Implementation.

```
int Insert_After(Node *head, int node_number, DATA_TYPE element){

    // linked list is empty
    if((head == NULL)){
        return 0;
    }

    // Invalid node number
    if((node_number > Length(head)) || (node_number < 1)){
        return -1;
    }

    // ------------------------------------------------

    Node *prev = head, *current;
    int x = 1;

    // create a node
    // reserve a block in heap
    current =(Node *)malloc(sizeof(Node));
```

Linked List Implementation.

```
    if(current == NULL){
        return 0;
    }

    while(x < node_number){
        prev = prev->next;
        x++;
    }

    // set data into a new node
    current->element = element;
    current->next = prev->next;

    prev->next = current;

    return 1;
}
```

electronics010.blogspot.com Basic Data Structures Eng. Mohamed Yousef

LAB

```
1   /*
2   ============================================================
    ====================
3   PROGRAM     : Linked list implementation
4   Author      : Eng. Mohamed Sayed Yousef
5                 http://electronics010.blogspot.com.eg/
6   Date        : November 2018
7   Version     : 1.0
8   Description :
9   ============================================================
    ====================
10  */
11  #include <stdio.h>
12  #include "lib/list.h"
13
14  //
    ============================================================
    ================
15
16  int main(){
17  Node *ptr_head = NULL;
18  int len = 0, x = 0;
19  DATA_TYPE element = 0;
20  DATA_TYPE *ptr_element = &element;
21
22      ptr_head = Append(ptr_head, 'S');
23      Append(ptr_head, 'y');
24      Append(ptr_head, 's');
25      Append(ptr_head, 't');
26      Append(ptr_head, 'e');
27      Append(ptr_head, 'm');
28
29      //
    ------------------------------------------------------
30
31      len = Length(ptr_head);
32      for(x=1; x<= len; ++x){
33          Get_Element(ptr_head, x, ptr_element);
34          printf("\n %d --> %c", x, *ptr_element);
35      }
36
37      printf("\n ---------");
38
39      //
    ------------------------------------------------------
40
41      ptr_head = Delete_First_Node(ptr_head);
42
43      len = Length(ptr_head);
44      for(x=1; x<= len; ++x){
45          Get_Element(ptr_head, x, ptr_element);
46          printf("\n %d --> %c", x, *ptr_element);
47      }
48
49      printf("\n ---------");
```

```
50
51      //
    ------------------------------------------------------
52
53      Delete_Node(ptr_head, 5);
54      len = Length(ptr_head);
55      for(x=1; x<= len; ++x){
56          Get_Element(ptr_head, x, ptr_element);
57          printf("\n %d --> %c", x, *ptr_element);
58      }
59
60      printf("\n ---------");
61
62      //
    ------------------------------------------------------
63
64      ptr_head = Insert_Frist(ptr_head, 'A');
65      len = Length(ptr_head);
66      for(x=1; x<= len; ++x){
67          Get_Element(ptr_head, x, ptr_element);
68          printf("\n %d --> %c", x, *ptr_element);
69      }
70
71      printf("\n ---------");
72
73      //
    ------------------------------------------------------
74      Insert_After(ptr_head, 2, 'M');
75      len = Length(ptr_head);
76      for(x=1; x<= len; ++x){
77          Get_Element(ptr_head, x, ptr_element);
78          printf("\n %d --> %c", x, *ptr_element);
79      }
80
81      printf("\n ---------");
82
83      //
    ------------------------------------------------------
84
85      return 0;
86  }
87
```

```
1   /*
2   ================================================================
    ==================
3   PROGRAM     : Linked list implementation
4   Author      : Eng. Mohamed Sayed Yousef
5                 http://electronics010.blogspot.com.eg/
6   Date        : November 2018
7   Version     : 1.0
8   Description :
9   ================================================================
    ==================
10  */
11  #ifndef LIST_H_INCLUDED
12  #define LIST_H_INCLUDED
13  // ------------------------------------------------
14
15  #define DATA_TYPE char
16
17  typedef struct node{
18      DATA_TYPE element;
19      struct node *next;
20  }Node;
21
22  // Prototypes
23  // ----------
24  Node* Append(Node *, DATA_TYPE);
25  int Length(Node *);
26  int Get_Element(Node *, int, DATA_TYPE *);
27  Node* Delete_First_Node(Node *);
28  int Delete_Node(Node *, int);
29  Node* Insert_Frist(Node *, DATA_TYPE);
30  int Insert_After(Node *, int, DATA_TYPE);
31
32  // ------------------------------------------------
33  #endif // LIST_H_INCLUDED
34
```

```
1   /*
2   ================================================================
    ==================
3   PROGRAM     : Linked list implementation
4   Author      : Eng. Mohamed Sayed Yousef
5                 http://electronics010.blogspot.com.eg/
6   Date        : November 2018
7   Version     : 1.0
8   Description :
9   ================================================================
    ==================
10  */
11  #include "list.h"
12  #include <stdlib.h>
13
14  //
    ================================================================
    ================
15  Node* Append (Node *head, DATA_TYPE element){
16  Node *new_node = NULL;    // Local (in stack)
17
18      // create a node
19      // reserve a block in heap
20      new_node =(Node *)malloc(sizeof(Node));
21
22      if(new_node == NULL){
23          return NULL;
24      }
25
26      // set data into a new node
27      new_node->element = element;
28      new_node->next = NULL;
29
30      // link the new node to the 'List'
31      if(head != NULL){ // linked list is not empty
32          // set address in head to 'temp'
33          // this address points to 1st node
34          Node *temp = head;
35
36          while(temp->next != NULL){
37              temp = temp->next;
38          }
39
40          temp->next = new_node;
41      }
42
43      // return address of new 'Node'
44      return new_node;
45  }
46  //
    ================================================================
    ================
47  int Length(Node *head){
48  int len = 1;
49
```

```
50      if(head == NULL){ // linked list is empty
51          return 0;
52      }
53      // set address in head to 'temp'
54      // this address points to 1st node
55      Node *temp = head;
56
57      while(temp->next != NULL){
58
59          len++;
60          // move temp to the next node
61          temp = temp->next;
62      }
63
64      return len;
65  }
66  //
    ================================================================
    ================
67  int Get_Element(Node *head, int node_number, DATA_TYPE
    *element){
68
69      if(head == NULL){ // linked list is empty
70          return 0;
71      }
72
73      if((node_number > Length(head)) || (node_number < 1)){
74          return -1;
75      }
76      // ------------------------------------------------
77
78      // set address in head to 'temp'
79      // this address points to 1st node
80      Node *temp = head;
81      int x = 1;
82
83      while(x < node_number){
84          // move temp to the next node
85          temp = temp->next;
86          x++;
87      }
88
89      *element = temp->element;
90
91      return 1;
92  }
93  //
    ================================================================
    ================
94  Node* Delete_First_Node(Node *head){
95
96      // linked list is empty
97      if(head == NULL){
98          return NULL;
99      }
```

```
100
101         // check List length
102         if(Length(head) == 1){
103             free(head);
104             return NULL;
105         } else { // more than one node
106             Node *second_node;
107             second_node = head->next;
108             free(head);
109             return second_node;
110         }
111     }
112     //
      ================================================================
      ================
113     int Delete_Node(Node *head, int node_number){
114
115         // linked list is empty
116         if((head == NULL)){
117             return 0;
118         }
119
120         // Invalid node number
121         if((node_number > Length(head)) || (node_number < 2)){
122             return -1;
123         }
124         // ------------------------------------------------
125
126         // delete any node in the middle or in last
127         Node *prev = head, *current;
128         int x = 1;
129
130         while(x < node_number - 1){
131             prev = prev->next;
132             x++;
133         }
134
135         current = prev->next;
136         prev->next = current->next;
137        // current->next = NULL;
138         free(current);
139
140         return 1;
141     }
142     //
      ================================================================
      ================
143     Node* Insert_Frist(Node *head, DATA_TYPE element){
144
145         // linked list is empty
146         if(head == NULL){
147             return NULL;
148         }
149
150         Node *first_node;
```

```
151
152         // create a node
153         // reserve a block in heap
154         first_node =(Node *)malloc(sizeof(Node));
155
156         if(first_node == NULL){
157             return head;
158         }
159
160         // set data into a new node
161         first_node->element = element;
162         first_node->next = head;
163
164         head = first_node;
165
166         return head;
167     }
168     //
    ============================================================
    ================
169     int Insert_After(Node *head, int node_number, DATA_TYPE
    element){
170
171         // linked list is empty
172         if((head == NULL)){
173             return 0;
174         }
175
176         // Invalid node number
177         if((node_number > Length(head)) || (node_number < 1)){
178             return -1;
179         }
180
181         // ------------------------------------------------
182
183         Node *prev = head, *current;
184         int x = 1;
185
186         // create a node
187         // reserve a block in heap
188         current =(Node *)malloc(sizeof(Node));
189
190         if(current == NULL){
191             return 0;
192         }
193
194         while(x < node_number){
195             prev = prev->next;
196             x++;
197         }
198
199         // set data into a new node
200         current->element = element;
201         current->next = prev->next;
202
```

```
203         prev->next = current;
204 
205         return 1;
206     }
207     //
        ================================================================
        ================
208 
```