Data Structure & Programming

DPP

Linked List

Section-01

[NAT]

1. Consider a single linked list q with 2023 elements is passed to the following function:

```
struct node {
   int data;
   struct node *next;
};
void f(struct node *q){
    struct node *p;
    p=q->next;
   q->next=p->next->next;
}
```

The size of the linked list q after the execution of the function is _____.

[MCQ]

2. Consider a single linked list q[‘A’, ‘B’, ‘C’, ‘D’, ‘E’, ‘F’] is passed to the following function:

```
struct node {
   int data;
   struct node *next;
};
void f(struct node *q)
{
    struct node *p;
    p=q->next->next->next;
    q->next->next->next=p->next->next;
    p->next->next=q->next;
    printf(“%c”, p->next->next->next->data);
}
```

The output is-

(a) C (b) D

(c) E (d) B

[NAT]

3. Consider the following statements:

P: Linked Lists supports linear accessing of elements

Q: Linked Lists supports random accessing of elements.

Which of the following statements is/are INCORRECT?

(a) P only (b) Q only

(c) Both P and Q (d) Neither P nor Q

[MCQ]

4. Consider a single linked list q[‘A’, ‘B’, ‘C’, ‘D’] is passed to the following function:

```
void f(struct node *q)
{
   if(q==NULL) return;
   f(q->next);
   printf(“%c ”, q->data);
}
```

The output is-

(a) C D B A (b) D C B A

(c) A B C D (d) B C D A

[NAT]

5. Consider the following statements:

P: Insertion at the end of the linked list is difficult than insertion at the beginning of the linked list.

Q: Deletion at the beginning of linked list is easier as compared to deletion at the end of the linked list.

Which of the following statements is/are CORRECT?

(a) Both P and Q (b) P only

(c) Q only (d) Neither P nor Q

[NAT]

6. The following C function takes a single-linked list p of integers as a parameter. It deletes the last element of the single linked list. Fill in the blank space in the code:

```
struct node {
   int data;
   struct node *next;
};
void delete_last(struct node *head)
{
   struct node *p=head, *q;
   if(!head) return;
   if(head->next==NULL){free(head);head=NULL;
   return;}
   while(______a_________){
      q = p;
      p=p->next;
   }
   ______b_________;
free(p);
 q=NULL; p=NULL;
}
```

(a) a: !head ; b: q->next = NULL;
(b) a: p->next ! = head ; b: q->next = q
(c) a: p->next ! = NULL ; b: q->next = NULL
(d) a: head->next ! = p ; b: q->next = p

[MCQ]

7. Consider a single linked list q[[‘A’, ‘B’, ‘C’, ‘D’, ‘E’, ‘F’, ‘G’] is passed to the following function:

```
void func(struct node *head){
   struct node *p=head, *q=head;

   while(q!=NULL && q->next!=NULL && q->next->next != NULL){
      p=p->next;
     q=q->next->next;
   }
   printf(“%c”, p->data);
}
```

The output is-

(a) C (b) D
(c) E (d) B

[NAT]

8. The following C function takes a single-linked list p of integers as a parameter. It inserts the element at the end of the single linked list. Fill in the blank space in the code:

```
struct node
{
   int data;
   struct node *next;
};
void insert_last(struct node *head, struct node *q){
struct node *p=head;
if(!head) return;
while(______a______){
   p=p->next;
   _____b_____;
   q=NULL;
   p=NULL;
  }
}
```

Assume, q is the address of the new node to be added.

(a) a: !head ; b: q->next = NULL;
(b) a: q->next ! = NULL; b: p->next = q
(c) a: p->next ! = NULL ; b: p->next = q
(d) a: head->next ! = p ; b: q->next = p

Answer Key

1. **(2021)**
2. **(a)**
3. **(b)**
4. **(b)**
5. **(a)**
6. **(c)**
7. **(b)**
8. **(c)**

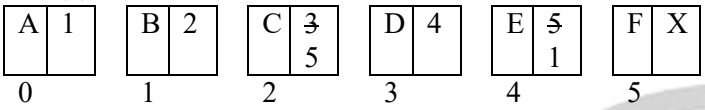
Hints and Solutions

1. (2021)

The above function implementation skip the second and third elements. It connects the head element to the fourth element.

So, the size of the linked list is 2021.

2. (a)



X represents NULL.
Initially, q points to node 0.

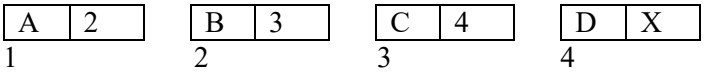
p=q->next->next->next;//p=3
q->next->next->next=p->next->next;//2->next=5
p->next->next=q->next;//4->next=1
printf("%c", p->next->next->next->data);
3->next->next->next->data
=4->next->next->data
=1->next->data
=2->data
=C

3. (b)

Linked List supports only linear accessing of elements.

4. (b)

```
void f(struct node *q){
if(q==NULL) return;
f(q->next);
printf("%c", q->data);
}
```



X represents NULL.
f(1):
1 is NOT NULL.
f(2).
P4: It prints 1->data i.e A.
f(2):
2 is NOT NULL.
f(3).
P3: It prints 2->data i.e B.
f(3):
3 is NOT NULL
f(X).
P2: It prints 3->data i.e C.
f(4):
4 is NOT NULL;
f(X).
P1: It prints 4->data i.e D.
f(X):
X is equal to NULL. So it returns to f(4);
OUTPUT: D C B A

5. (a)

P: CORRECT. Insertion at the end of the linked list is difficult than insertion at the beginning of the linked list.

Q: CORRECT. Deletion at the beginning of linked list is easier as compared to deletion at the end of the linked list.

6. (c)

```
void delete_last(struct node *head)
{
    struct node *p=head, *q;
  if(!head) return;
  if(head->next==NULL){free(head);head=NULL;
   return;
}
while(p->next!=NULL)
{
    q = p;
    p=p->next;
 }
q->next=NULL;
free(p);
 q=NULL; p=NULL;
}
```

7. (b)

The code prints the middle element in the linked list q.
A B C D E F G.
Output: D

8. (c)

```
void insert_last(struct node *head, struct node *q){
struct node *p=head;
if(!head) return;
while(p->next!=NULL)
p=p->next;
p->next=q;
q=NULL;
p=NULL;
}
```

CSE/IT Batch-English

Data Structure & Programming
Linked List

Section-02

[NAT]

1. Consider a linked list [a, b, c, d, e]. ptr is a pointer pointing to the head/start node. Assume a node in the linked list is defined as:

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

The output of the statement printf("%d", ptr->next->next->data) is ____.

[NAT]

2. Consider a single linked list of integers [9, 8, 7, 6, 5, 4,3] is passed to the following function:

```
struct node
{
int data;
struct node *next;
};
int  func(struct node *q){
static int k=0;
struct node *ptr=q;
if(!ptr) return 0;
else if(ptr->next==NULL) return k+=ptr->data;
else{
k+=ptr->data;
 func(ptr->next);
return k;
}
}
```

Assume, q points to head/start node in the linked list. The value returned by func(q) is _____.

[MCQ]

3. Consider the following function:

```
struct node
{
int data;
struct node *next;
};
struct node * f(struct node *head, int k){
struct node *p=head;
int i=0;
while(i<k/2){
p=p->next;
i++;
}
return p;
}
```

Assume head points to the start node of the linked list and k is the number of elements in the linked list, the function returns-

(a) The pointer to the middle element in the linked list.
(b) The pointer to the second element in the linked list.
(c) The middle element in the linked list.
(d) The second last element in the linked list.

[MCQ]

4. Consider the following function:

```
struct node
{
int data;
struct node *next;
};
void f(struct node *head){
struct node *a=head, *b=NULL, *c=NULL ;
while(a){
c=a->next;
a->next=b;
b=a;
a=c;
}
head=b;
}
```

Assume head points to the start node of the linked list, the function-

(a) Sorts the linked list.
(b) Interchanges every two consecutive elements in the list.

(c) Reverses the list.
(d) None of the above.

[MCQ]

5. Consider a single linked list [1, 2, 3, 4, 5] is passed to the following function:

```
struct node
{
int data;
struct node *next;
};
void  func(struct node *p){
struct node *q=p->next, *temp;
if(!p||!(p->next)) return;
else{
temp=q->data;
q->data=p->data;
p->data=temp;
func(p->next->next);
}
}
```

Initially, the address of the head/start node is passed to the function func(*p), the arrangement of the linked list after function execution is –

(a) 2 3 4 5 1 (b) 5 4 3 2 1
(c) 2 1 4 3 5 (d) 2 1 4 5 3

[MSQ]

6. Consider the following function:

```
struct node
{
int data;
struct node *next;
};
struct node * f(struct node *head){
struct node *p=head, *q=head;
while(q!=NULL && q->next!=NULL && q->next->next != NULL){
p=p->next;
q=q->next->next;
if(p==q) break;
}
return p;}
```

Assume head points to the start node of the linked list, the function-

(a) Returns the pointer to the node where a cycle/loop ends (assume the leftmost node in a loop is the start node of a cycle).
(b) Returns the pointer to the node where a cycle/loop starts (assume the leftmost node in a loop is the start of a cycle).
(c) Reverses the list.
(d) Detects a cycle in the list.

[MCQ]

7. Consider the following function:

```
struct node
{
int data;
struct node *next;
};
void f(struct node *head, int e){
struct node *p, *q;
if(head->data==e){
q=p;
p=p->next;
_________;
head=p;
return;
}
q=head; p=head->next;
while(p->next!=NULL){
if(p->data==e){
_____________;
free(p);
return;
}
q=p;
p=p->next;
}
if(p->data==e){
q->next=NULL;
free(p);
}
}
```

Assume there are at least two elements in the single linked list of integers. The starting node's address is contained in the head pointer passed to the function. The function f() searches for the element e in the list. If found, the function deletes the node. The missing statements are-

(a) free(q), q->next=p->next
(b) free(q), q=p->next
(c) free(p), q=p->next
(d) free(p), q->next=p->next

[MCQ]

8. A node of a linked list is to be created by calling malloc() function. The malloc() returns NULL if-

(a) Stack overflow occurs.

(b) Memory leakage occurs.

(c) Memory is full.

(d) None of the above.

Answer Key

1. **(99)**
2. **(42)**
3. **(a)**
4. **(c)**
5. **(c)**
6. **(b, d)**
7. **(a)**
8. **(c)**

Hints and Solutions

1. (99)

printf("%d", ptr->next->next->data); //The ASCII of c is printed i.e 99

2. (42)

It computes the sum of all the data elements in a recursive manner.

Output = 9+8+7+6+5+4+3=42

3. (a)

It returns the pointer to the middle element in the linked list.

4. (c)

It reverses the list.

5. (c)

The function reverses the elements in groups of 2.

So, output is 2 1 4 3 5

6. (b, d)

It detects a cycle in the linked list and returns the pointer to the node where the loop starts from.

7. (a)

```
void f(struct node *head, int e){
struct node *p, *q;
if(head->data==e){
q=p;
p=p->next;
free(q);
head=p;
return;
}
q=head; p=head->next;
while(p->next!=NULL){
if(p->data==e){
q->next=p->next;
free(p);
return;
}
q=p;
p=p->next;
}
if(p->data==e){
q->next=NULL;
free(p);
}
}
```

8. (c)

The malloc() returns NULL only if the memory is full.