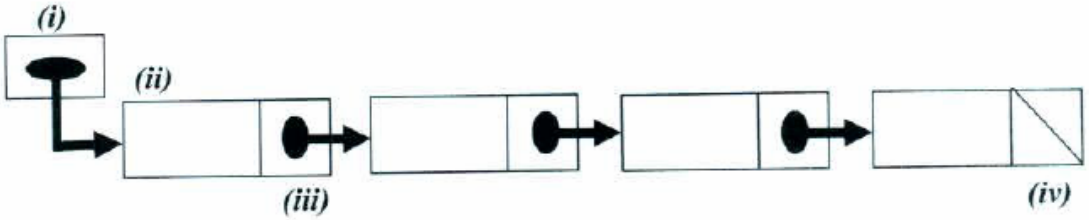
Chapter 3 (Linked list)

April 04

A2. Consider the diagram of a linked list below, and answer the questions which follow it.



(a) Identify the areas labelled (i) to (iv). [4]

(b) (i) How many nodes are there? [1]

(ii) If the first node is referred to with *Temp*, and the second node with *Temp->Link*, then how should the very last node be referred to? [2]

Answer:

(a)
(i) Head [1]
(ii) Information/Data field [1]
(iii) Pointer field [1]
(iv) NULL [1]
[max 4]

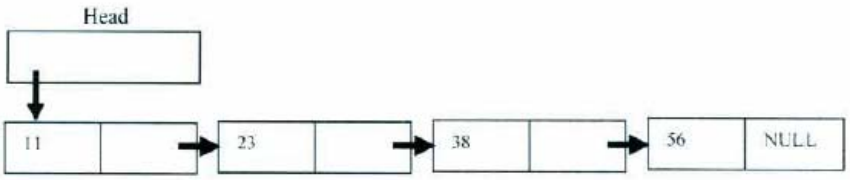
(b)
(i) 4 nodes [1]
(ii) Temp->Link->Link->Link [2]
[max 3]

April 04

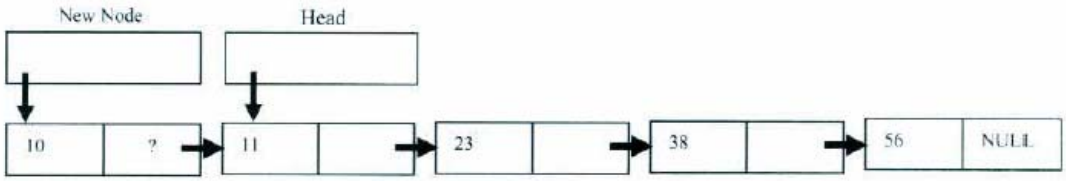
B2

(b) Based on the diagram below, answer questions (i) to (iv).

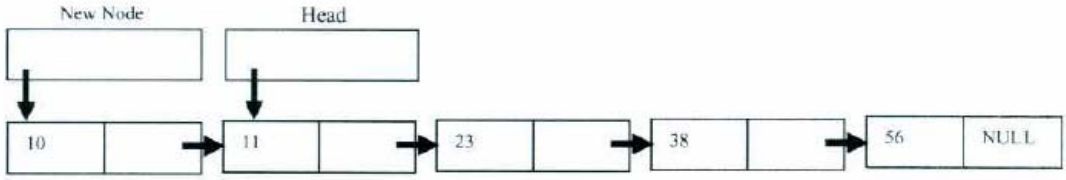
1.



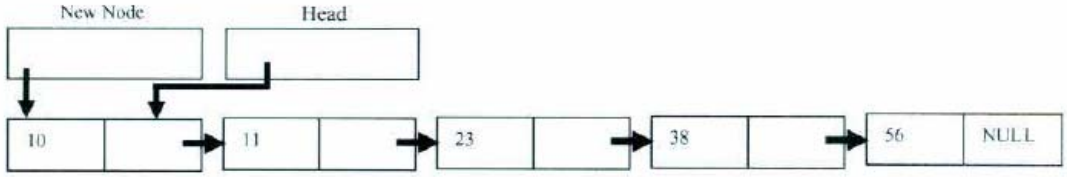
2.



3.



4.



Assume this is the structure of the list.

```
struct Node{
  int Data;
  struct Node *Link;
  };
```

(i) Write C code based on the diagrams shown to represent the algorithm for adding a node to the front of a linked list. Your function should have the following header. [10]

```
void InsertInFront (NodePointer &Head, int Number)
{
.....
}
```

Answer:

```
void InsertInFront(NodePointer &Head,int Number)
{
        NodePointer NewNode;                          [1]
        NewNode = (NodePointer)malloc(sizeof(struct
        Node));                                       [3]
        NewNode->Data = Number;                       [2]
        NewNode->Link = Head;                         [2]
        Head = NewNode;                               [2]
}                                                  [max 10]
```

(ii) Explain the purpose of NULL in the diagrams of linked lists. [2]

Answer:

NULL in the diagram is to show that the last node does not point to anything. [1]
It is the end of the list. [1]
[max 2]

(iii) What would happen if, in diagram 4, Head is assigned NULL? Eg. Head = NULL; [4]

Answer:

The list would become an empty list. [1]
The rest of the nodes that were pointed to by Head would become lost nodes. [1]
They become useless memory locations which cannot be reused, [1]
known as garbage. [1]
[max 4]

(iv) What is the necessary check that should be made to avoid the problem discussed in (iii)? [2]

Answer:

Need to check whether the list is empty or not before assigning NULL to head. [2]
[max 2]

December_03

B2. (c) The steps below show how to insert a Node at the end of a linked list.

- Create a new node pointed to by NewNode.
- Make Temp point to Head
- Move Temp to the last node
- Make Temp's link point to NewNode

(i) Using the outline above, complete the following program by adding variables or function or values, so that the algorithm will insert a Node at the end of the list. [7]

You may assume the following declarations:

```
struct Node {
   int Info;
   struct Node *Link;
};

typedef struct Node *Pointer;
```

The program to be completed is:

```
void InsertAtEnd ( Pointer &Head, int value) {
   Pointer ____1_____, Temp;
   NewNode = (______2____) malloc ( ____3____ (struct Node);
   NewNode->Info = value;
   NewNode->Link = ___4___;

   if ( Head == NULL)
      Head = NewNode;
   else {
      Temp = __5_____;
      while (Temp->Link!=NULL)
         Temp = _____6____;
         Temp-> Link = _____7_____;
   }
}
```

1. **NewNode [1]**
2. **Pointer [1]**
3. **sizeof [1]**
4. **NULL [1]**
5. **Head [1]**
6. **Temp->Link [1]**
7. **NewNode [1]**

(ii) Explain the purpose of the statements shown in italic above in part (i).

1. if (Head == NULL) [1]
   Head = NewNode; [1]

   Answer:

   Is to check whether is list is empty. [1]
   If empty, assign the new node at the first node (Head) [1]

2. while (Temp->Link!=NULL) [1]

   Answer: Any of the following, for 1 mark

   Check whether is it at the end of list [1]

   or

   Move Temp to the last node. [1]

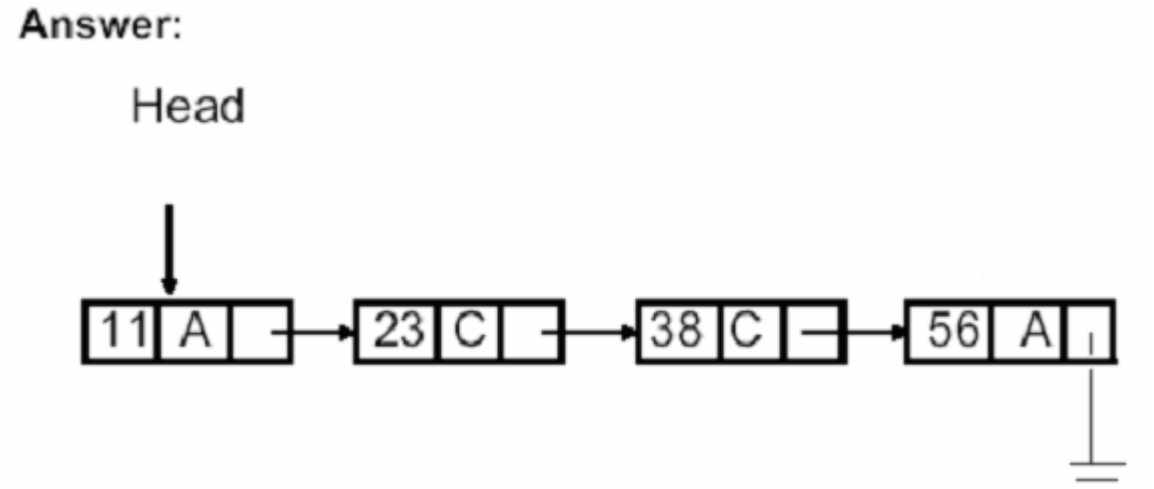
August_03

B2. (a) (i) Explain the features of a Linked List. [6]

- **A linked list is a dynamic data structure [1]**
- **Each node in the list contains at least two fields. [1]**
- **One is data field which store data item [1]**
- **The other one is pointer field which contain the address of the next node in the list. [1]**
- **The successor of each node is the next node in the list. [1]**
- **The pointer field in the last node of the list does not point to anything. So it is given a NULL value. [1]**

(ii) Draw an example of a linked list, based on what you explained in part a(i) above. The list should contain 4 nodes, with each node containing 2 information fields. [6]

Answer:



1 mark for showing the Head
1 mark for showing 4 nodes
2 mark for showing 2 information fields and
1 mark for showing the address field
1 mark for showing the last node with a NULL pointer.

(iii) Linked list structures usually save space, time and programming effort. Explain two circumstances where it is appropriate to use a linked list. [2]

Answer: Any 2 of the following, 1 mark each.

- **It needs to hold large records [1]**
- **The size of the list is not known in advance [1]**
- **Flexibility is needed during insertions, deletions and reordering of elements. [1]**

April_03

A6. Consider the algorithm for deleting a node from a linked list.

(a) Is deletion of the last node of a linked list a special case? Explain your answer. [2]

No [1]
If cur points to the last node and prev points to the next-to-last node, then prev→next=cur→next sets the next pointer of the next-to-last node to NULL, as required. [1]

(b) Is deletion of the only node of a one-node linked list a special case? Explain your answer. [2]

No [1]

This case is the same as deletion of the first node [1]

(c) Does deleting the first node take more effort than deleting the last node? Explain your answer. [4]

- **Locating the last node requires a traversal of the whole list [1]**
- **So it takes more effort than locating the first node. [1]**

- **However, deleting the last node is easier than deleting the first node [1] because you do not need to change head. [1]**

April_03

B2 (a) Write a C function to display the ith integer in a linked list of integers. Assume that i>=1 and that the linked list contains at least i nodes.

The function should have the following header:

void display (Node *head, int i)

and you may assume the following type definitions:

```
struct list {
   int item;
   struct list *next;
};

typedef struct list *Node; [6]
```

Suggested Answer

```
void display ( Node *head, int i) {
   Node *cur=head; [1]

   for ( int count = 1; count < i ; ++count) {  [2]
      cur = cur ->next; [1]
   }
   printf("%d", cur->item); [2]
}
```

August_04

B2. (a) (i) Define the term Linked List. How are the list nodes connected? [2]

A linked list is a dynamic variable that consists of nodes containing valuable data [1]

each node in the linked list is connected by pointers [1]

(ii) What is meant by the term 'Empty List'? [3]

An 'Empty List' means nothing inside the linked list. [1]

The head pointer which point to the first node of the list is equal to NULL. [1]

i.e. Head = NULL; [1]

(iii) Give the C statements to define a linked list, where each node contains the following fields. You may use any type or field names you want. [7]

- Name, eg: Cally
- Age, eg: 50
- Gender, eg; 'F' or 'M'
- a next address (point to the next record)

```
struct Information {  [1]
    char Name[20]; [1]
     char gender; [1]
     int age; [1]
     struct Information *next; [1]
};

typedef struct Information *Student; [1]

void main ( ) {
   Student Head; [1]
}
```

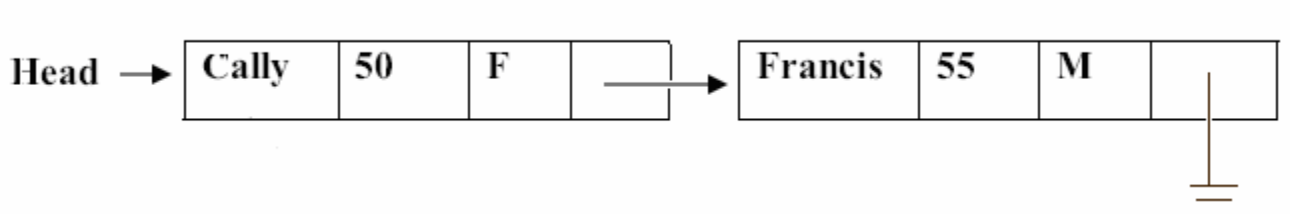
(iv) Draw a diagram to represent a linked list as declared in (a)(iii), where the first and second nodes contain the following information: [5]

1st node

Name	Cally
Age	50
Gender	F

2nd Node

Name	Francis
Age	55
Gender	M



1 mark for Head pointer
1 mark for correct data fields in 1st node
1 mark for pointer to 2nd node
1 mark for correct data fields in 2nd node
1 mark for null pointer

December_01

A2. Explain the effect of each of the following statements:

a) Head = Head->Link; [1]
b) printf("%p", Head); [1]
c) free(Head); [1]

A2. a) Move pointer Head to the next node [1]
b) Display the value of Head [1]
c) Dispose the memory location pointed by Head [1]

April_02

A2. Explain what happens when each of the following statements is executed:

(a) While (Head->Link != NULL)
Head = Head->Link; [1]

(b) free(Head); [1]

(a) Check for empty List; if not loop continues by moving pointer Head to the next node
[1]
[Accept also: make pointer Head refer to the last node]

(b) Dispose / free the memory location pointed to by Head [1]

April_02

A3. Explain the meaning of each of the following terms:

(c) Linked List [2]

(c) A linked list is a dynamic variable that consists of nodes containing data [1]
Nodes in a linked list are connected by pointers. [1]

December_00

B2. (b) Consider the following type definition:

```
typedef struct node {
        int   number;
        char grade;
}*NODEPTR;
```

Why can't the above declaration not be used for a node in a linked list? What should be done to rectify this problem? [3]

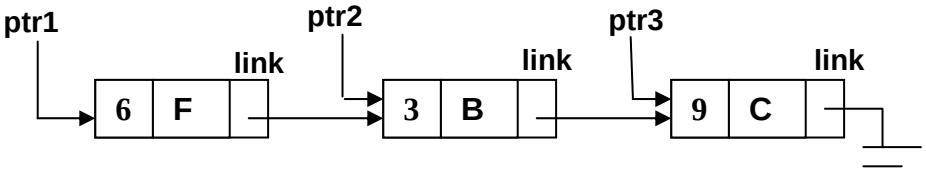
Suggested answer

The node does not have a pointer field. [1]

The correct one should be:

```
typedef struct node {
        int   number;
        char grade;
        struct  node  *link;
}*NODEPTR;
```
[2]

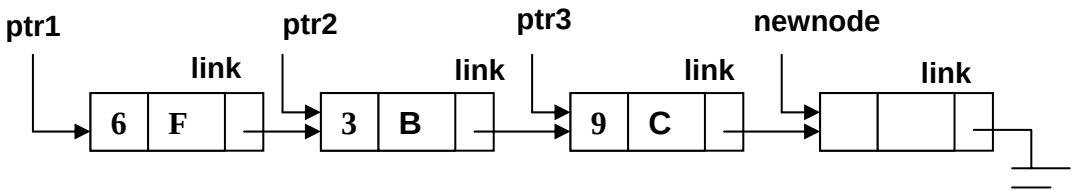
(c) With reference to the linked list below, give C statements to create a new empty node called *newnode* and add it to the back of the list. [3]



Suggested answer

NODEPTR newnode = (NODEPTR) malloc(sizeof(struct node)); [1]
ptr3->link = newnode; [1]
newnode->link = NULL; [1]

(d) Assume that after adding the new node, the linked list looks like this:



Give the C code to show how you would delete the second node without causing other nodes to be lost. [3]

Suggested answer

ptr1->link = ptr3; [2]
free(ptr2); [1]

(e) Give the C statement to print the grade in the third node by using the pointer ptr1. [2]

Suggested answer

printf("%c", ptr1->link->link->grade); [2]

(f) Suppose that *head* points to the first node in a linked list. Which node does the following expression refer to?

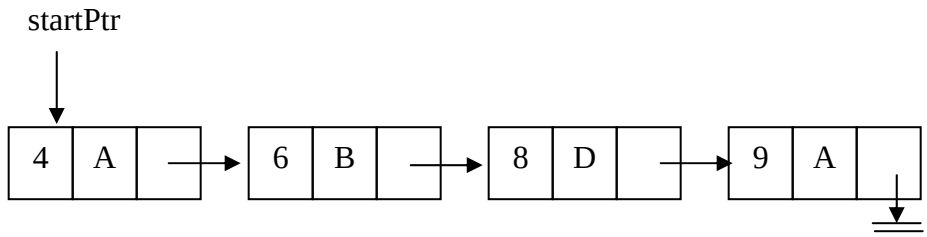
head->link->link->link->link->link; [1]

Suggested answer

the sixth (6 th) node. [1]

April_01

B1. (a) The linked list below has 4 nodes and a pointer variable called *startPtr* that points to the first node of the list. Answer the questions below based on the linked list drawn.:-



(i)
```
struct Node
{
    _______ ________;
    _______ ________;
    _______ ________ ________;
}

typedef  _________ _________ *NodePointer;

_________ startPtr;
```

Complete the declaration above, which is for the linked list drawn. [8]

Suggested answer

```
struct Node {
    int  number;                    [1]
    char  grade;                    [1]
    struct Node *next;              [3]
}

typedef struct Node *NodePointer;   [2]

NodePointer startPtr;               [1]
```

(ii) Declare a pointer called *CurrentPtr* of the same type as *startPtr*. Move *CurrentPtr* to the end of the list. [2]

Suggested answer

```
NodePointer CurrentPtr;                                    [1]
CurrentPtr = startPtr->next->next->next;                   [1]

OR

NodePointer CurrentPtr;                                    [1]
CurrentPtr = startPtr;
while (CurrentPtr->next != NULL)                           [1]
    CurrentPtr = CurrentPtr->Next;
```

(iii) Assuming that pointers *NewPointer, BeforePtr* and *AfterPtr* have also been declared, make *NewPointer* point to a new node and assign the value 7 and 'C' into the respective fields in the node.

Make *BeforePtr* point to the second node and *AfterPtr* point to the third node. Then insert the new node in between these two nodes.

Write C statements to perform all the tasks above. You should describe the fields of the nodes using the names declared in part (i). [7]

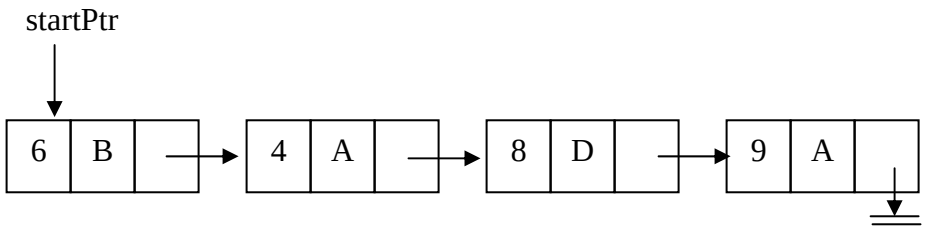
Suggested answer

```
NewPointer = (NodePointer)malloc(sizeof(struct Node));  [1]
NewPointer ->number = 7;                                [1]
NewPointer->grade = 'C';                                [1]
BeforePtr = startPtr->next;                             [1]

AfterPtr = startPtr->next->next; OR
AfterPtr = BeforePtr->next;                             [1]

BeforePtr->next = NewPointer;                           [1]
NewPointer->next = AfterPtr;                            [1]
```

(iv) Write a C function to interchange the first two nodes in the linked list. For instance, after the function has been executed, the original linked list above should look as follows: [5]



Use the following function header :-

```
void Interchange(NodePointer *startPtr)
```

Suggested answer

```
void Interchange(NodePointer *startPtr)
{
    NodePointer temp;                          [1]
    temp = startPtr->next;                     [1]
    startPtr->next = temp->next;               [1]
    temp->next = startPtr;                     [1]
    startPtr = temp;                           [1]
}
```

August_01

B1. (a) Consider the following declarations:

```
struct aNode
{
        int data;
        struct aNode *link;
};

typedef struct aNode *aNodePointer;
```

Write C statements for questions (i) to (iv) below, based on the declarations above.

(i) Declare **two** pointers of type aNodePointer and name them PointerX and PointerY. **[2 marks]**

Suggested answer

```
aNodePointer PointerX, PointerY;                    [2]
```

(ii) Using the *malloc* function, make PointerX point to a new memory location, store the value 20 in the new location and make its link point to NULL **[4 marks]**

Suggested answer

```
PointerX = (aNodePointer)malloc(sizeof(struct aNode));   [2]
PointerX→data = 20;                                      [1]
PointerX→link = NULL;                                    [1]
```

(iii) Make PointerY point to the same location as PointerX. **[2 marks]**

Suggested answer

```
PointerY = PointerX;                                [2]
```

(iv) Dispose the node that PointerY is pointing to. **[1 mark]**

Suggested answer

```
free(PointerY);                                     [1]
```

(v) What happens when the following statement is then executed? **[1 mark]**
printf(PointerX->data);

Suggested answer

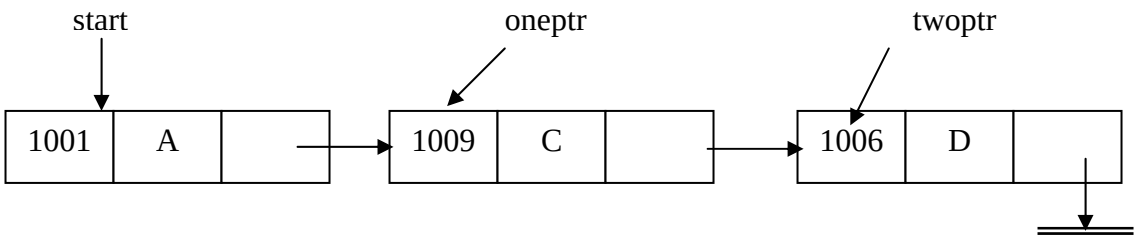
A run time error occurs **[1]**

Remark: free (PointerY) in part (iv) means disposing the memory location pointed by PointerY.

Because PointerX and PointerY are pointing to the same location (refer to part (iii)), free PointerY also makes PointerX undefined.

December_01 (Linked List)

B3. b) Below is an example of a linked list consisting of 3 nodes and 3 pointer variables. The nodes and the pointer variables are declared below.




```
struct studentnode
{
        int studentid;
        char grade;
        struct studentnode *link;
};
typedef struct studentnode *studentpointer;
studentpointer start, oneptr, twoptr;
```

i) The list above is not in proper ascending order. The node with the studentid 1006 must be placed between 1001 and 1009. Write C statements to exchange the nodes. You should use only the three pointers shown above and no other pointers. Make sure that the last node's link is NULL to indicate the end of the list. [3]

Suggested answer

start->link = twoptr; **[1]**
twoptr->link = oneptr; **[1]**
oneptr->link = NULL; **[1]**

ii) Assume now that the nodes have been interchanged, with pointer *oneptr* still pointing to node 1009 and *twoptr* pointing to node 1006. Using another pointer, called *newptr*, make *newptr* point to a new location. Store the value 1005 in member *studentid* and 'A' in member *grade*, in the location pointed to by *newptr*. [4]

Suggested answer

```
studentpointer newptr;
newptr = (studentpointer)malloc(sizeof(struct studentnode));
                                                        [2]
newptr->studentid = 1005;                               [1]
newptr->grade = 'A';                                    [1]
```

iii) Insert the new location at the proper place in the list, making sure that the list is kept in ascending order. [3]

Suggested answer

```
start->link = newptr;                                   [2]
newptr->link = twoptr;                                  [1]
```

iv) With the inserted new node and *newptr* pointing at the new location, delete the third node of the list. Make sure all the nodes in the list are still linked correctly. [3]

Suggested answer

```
newptr->link = oneptr;   OR   newptr->link = twoptr->link; [2]
free(twoptr);[1]
```

April_02

B2. (b) Explain the steps in the algorithm to insert a node at the front of a Linked List. [5]

Suggested answer

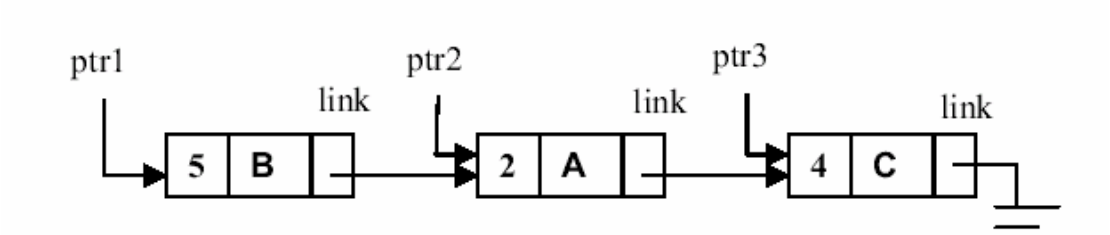
Create a new memory location pointed to by *NewNode* [1]
Place the data into the data field of the new node [1]
Make NewNode's link point to the head node of the original linked list . [2]
Make head point to the node that NewNode is pointing to [1]

August_02

B1. (f) Consider the following definitions

```
struct node {
   int number;
   char quality;
   struct node *link;
};
typedef struct node *Nodeptr;
```

(i) For the linked list below, give C statements to create a new empty node called

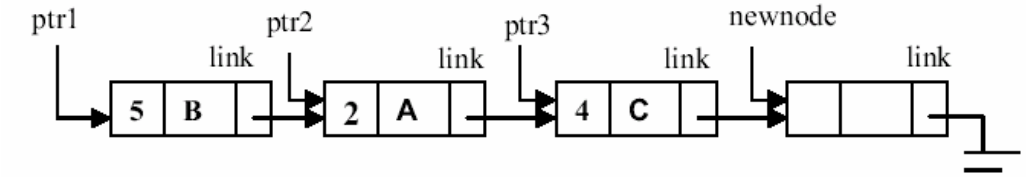


newnode and add it to the back of the list. [3]

Suggested answer

Nodeptr newnode = (Nodeptr) malloc(sizeof(struct node)); [1]
ptr3->link = newnode; [1]
newnode->link = NULL; [1]

(ii) Assume that after adding the new node, the linked list looks as follows:



Give the C code to show how you would delete the second node without causing other nodes to be lost. [3]

Suggested answer

ptr1->link = ptr3; [2]
free(ptr2); [1]

(iii) Give the C statement to print the number stored in the second node by using the pointer ptr1. [1]

Suggested answer

printf(“%d”, ptr1->link->number); [1]