UNIT-3

1. Discuss the differences between SMA and DMA.
2. Differentiate between arrays and linked list.

A.

Arrays	Linked List
1. It uses the static memory allocation	1. It uses the dynamic memory allocation
2. Memory is allocated at compile time	2. Memory is allocated at run time
3. The elements are stored in continuous memory location	3. The elements are stored in any available memory locations using pointers
4. The accessing is random	4. The accessing is sequential
5. The insertion and deletion from any position is time consuming	5. The insertion and deletion is quite easy

3. Explain calloc function. Write its syntax. write a C program to implement calloc function.

A. calloc or contiguous allocation method in C is used to dynamically allocate the specified number of blocks of memory of the specified type. It initializes each block with a default value of '0'

Syntax: ptr = (cast_type *) calloc (n, element_size);

ex: ptr = (float *) calloc (25, sizeof (float));

Program:

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
  int *ptr;
  int n, i;
  printf("\n enter no. of elements");
  scanf("%d", &n);
  ptr = (int *) calloc(n, sizeof(int));
  if (ptr == NULL)
  {
    printf ("\n memory not allocated");
    exit(0);
  }
  else
  {
  printf("\n memory successfully
        allocated using calloc");
  for (i=0; i<n; i++)
    ptr[i] = i+1;
  printf ("\n The elements of
        the array are ");
  for (i=0; i<n; i++)
    printf ("%d ", ptr[i]);
  }
  return 0;
}
```

4 Explain malloc function? Write its syntax. Write a C program to implement malloc function.

A malloc or memory allocation method in C is used to dynamically allocate a single large block of memory with the specified size.

It returns a pointer of type void which can be cast into a pointer of any form. It initializes each block with default garbage value.

Syntax: ptr = (cast-type *) malloc (byte-size)

ex: ptr = (int *) malloc (100 * sizeof(int));

Program:

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
  int *ptr;
  int n, i;
  printf("\n enter no. of. elements");
  scanf("%d", &n);
  ptr = (int *) malloc (n * sizeof(int));
  If (ptr == NULL)
  {
   printf("\n Memory not allocated");
   exit(0);
  }
  else
  {
  printf("\n Memory successfully
          allocated using malloc");
  for (i=0; i<n; i++)
     ptr[i] = i+1;
  printf("\n The elements of the
          array are: ");
  for (i=0; i<n; i++)
     printf("%d \t", ptr[i]);
  }
  return 0;
}
```

5. Can you compare the efficiency of linked lists with other data structures like arrays for various operations

A. Same answer (linked list Vs Arrays)

6. How do linked lists support dynamic memory allocation and deallocation.

A. Refer answer 3 & 4 — for memory allocation.

free method in C is used to dynamically de-allocate the memory. The memory allocated using functions malloc() and calloc() is not de-allocated on their own. Hence the free() method is used, whenever the dynamic memory allocation takes place. It helps to reduce wastage of memory by freeing it.

Syntax: free(ptr);

Program:

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
  int *ptr, *ptr1;
```

7. Provide code examples illustrating the implementation of basic operations such as insertion, deletion, traversal etc.

A program:

```
#include <stdio.h>
#include <stdlib.h>
Struct node
{
    int data;
    Struct node *next;
};
Struct node *head;
void Insert();
void del();
void display();
~~void search();~~
~~void reve~~
int c=0, b=0;
void main()
{
    int a, ch;
    head = NULL;
    while(1)
    {
      Scanf("%d", &ch);
      switch(ch)
      {
        case 1: Insert();
                break;
        case 2: del();
                break;
        case 3: display();
                break;
        case 4: exit(0);
        default: printf("Invalid
                choice");
      }
    }
}
void Insert()
{
    int p, i, n;
    Struct node *temp=(Struct
      node*)malloc(sizeof(Struct node));
    Struct node *temp2;
    Scanf("%d %d", &p, &n);
  if (p<=0 || p>c+1)
    printf("In position doesnot exist
        cannot insert into SLL");
```

```
int n, i ;
printf ("\n enter the no of elements");
scanf ("%d", &n);
ptr = (int *) malloc (n * sizeof (int));
ptr1 = (int *) calloc (n, sizeof (int));
if (ptr == NULL || ptr1 == NULL)
{
  printf ("\n memory not allocated");
  exit (0);
}
else
{
  printf ("\n memory successfully allocated using malloc");
  free (ptr);
  printf ("\n malloc memory successfully freed");
  printf ("\n memory successfully allocated using calloc");
  free (ptr1);
  printf ("\n calloc memory successfully freed");
}
return 0;
}
```

```
else
{
   c++;
   temp->data = n;
   temp -> next = NULL;
   if (p==1)
   {
     temp -> next = head;
     head = temp;
   }
   else
   {
     temp2 = head;
     for (i=0; i<p-2; i++)
       temp2 = temp2 -> next;
     temp -> next = temp2 -> next;
     temp2 -> next = temp;
   }
  }
}
void del()
{
    int p, i;
    struct node *temp1 = head,
                *temp2;
    scanf("%d", &p);
    if (p<=0 || p>c)
      printf("\n position does not
        exist - cannot delete from
        SLL");
    else
    {
      c--;
      if (p==1)
      {
        head = temp1 -> next;
        printf("\n Deleted element
          from SLL is %d", temp1->data);
        free(temp1);
      }
      else
      {
        for (i=0; i<p-2; i++)
          temp1 = temp1 -> next;
        temp2 = temp1 -> next;
        temp1 -> next = temp2 -> next;
        printf("\n Deleted element
          from SLL is %d", temp2->data);
      }
    }
```

```
void display()
{
  struct node *x;
  x = head;
  printf("\n");
  while (x != NULL)
  {
    printf("%d ->", x->data);
    x = x->next;
  }
  if (head == NULL)
    printf("Empty SLL - cannot display");
}
```

8. Explain different approaches and algorithms to reverse the elements of a single linked list and to remove duplicate elements form an unsorted SLL.

8. Explain different approaches and algorithms to reverse the elements of a SLL and to remove duplicate elements from an unsorted SLL.

A Algorithm to reverse the elements of SLL

Step 1: Initialize *P = NULL and assign x = head

Step 2: Check whether head is NULL or not. if so return else goto step 3.

Step 3: Repeat step 4,5 until x is NULL.

Step 4: Create newnode for x->data and store in y.

Step 5: 5.1. y->next = p;
5.2. P = y;
5.3 x = x -> next

Step 6: Store p in head and then display.

Algorithm to remove duplicate elements of unsorted SLL

There are two methods to do this

① Brute-force Approach - using two loops

② Optimized Approach - using Hashing

① using two loops

Step 1: At the initial step a linked list is created using the append() function. If the linked list is empty, then make a new node in the head, else add a new node after the last node.

Step 2 : create a function remove-duplicates() that accepts one parameter - the head pointer of the linked list.

Step 3 : Initialize two variables ptr1 and ptr2.

Step 4 : Set ptr1 = head and ptr2 to null

Step 5 : use while loop and stop when ptr1 of ptr1 → next is equal to NULL

Step 6 : Set ptr2 = ptr1

Step 7 : A nested while loop is used to make another iteration which terminates when the value of ptr2 → next is NULL.

Step 8 : If ptr1 & ptr2 are equal delete the node and increment ptr2 → next to ptr2 → next → next. If not equal, then increment ptr2 to its next node.

Step 9 : Increment ptr1 to its next node

Step 10 : Finally, use the printLinkedList() function