record[0].name =
record[1].gpa =

Structures
union
list
priority queue
Dynamic memory allocation
memory leak
Array of structures
Array of pointers
Parameter passing
Bitfields
enums.

DATE

Unit - 3.

Structure

```
struct student
{
    char srn;    }
    char name;   } structure members
    float gpa;   }
};
```

OR.

```
struct student
{
    char srn;
    char name;
    float gpa;
} S1, S2;
```

```
struct student S1, S2, S3;
```

Access structure members in 2 ways.
1. '.' operator
2. '->' operator.

```
S1.name;  // Accessing name of 1st member.
```

Array of structures

```
struct student record [4];
```

Structure can be passed to a function in 3 ways
1) By value
2) By address
3) Declare structure variable as global.

int → 4
int → 4
float → 4
char → 4.
16.

```
struct employee
{
    char ename;
    char enum;
    int age;     → 4
}   float salary → 4.
```

Memory allocation
ename / enum | a | l | | |
age | | | | |

PES UNIVERSITY

DATE

2 consecutive chars, then both 4 bytes total

All occupy 4 bytes but if there are consecutive char, then both occupy same, except double which occupies 8.

Q. Write a program to find the generic root of a given number. The generic root has to be found out for integers only and for integers greater than zero

4 5 6

→ 4+5+6 = 15 = 1+5 = 6

```
int main()
{
    int n;
    printf("Enter a number: ");
    scanf("%d", &n);
    (n%9==0)? printf("9"): printf("%d", n%9);
}
```

Q. WAP to print atleast 4 perfect numbers starting from 0

```
int main()
{
int n=1, i, j=0, k, sum=0;
while(j<4)
{
    for(i=1; i<=n; i++)
    {
    sum=0;
    for(k=1; k<i; k++)
        if(i%k==0)
            sum+=k
    n++;
```

DATE

```
        if (sum==i)
        {
          printf("%d, ",i);
          j++;
        }
      if(j==4)
        break;
      }
    }
}
```

Q. WAP to print strong numbers from 1 to n.

```
int main()
{
  int n,i,j,sum,dig,m,fact;
  printf("Enter n: ");
  scanf("%d",&n);
  for(i=0;i<=n;i++)
  {
    j=i;
    sum=0;
    while(j>0)
    {
      dig=j%10;
      fact=1;
      for(m=1;m<=dig;m++)
        fact=fact*m;
      j=j/10;
      sum+=fact;
    }
    if(sum==i)
      printf("%d\n",i);
  }
}
```

DATE

Unions:

```
Union student
{
    char name[30];
    int age;
    float gpa;
};
```

→ 30 bytes total, age & gpa get overwritten on 30 bytes

Only one difference from struct:
Diffis in terms of memory used.

Union — Whichever member occupies maximum bytes, that much memory will be allocated

Structure padding: In order to align the data in memory, one or more empty bytes are inserted or left empty between memory addresses which are allocated for other structure members while allocating memory. This is called structure padding
Architecture of comp. processor is such that it can read one word (4 bytes in 32 bit processor) from memory at a time.
To make use of this ~~advantage~~ architecture of a processor, data are always aligned as 4 bit package which leads to inserting empty adresses between other member adresses.

Command #pragma pack(1). → no empty spaces in memory for

Union is similar to structure, each element in union is called a member

→ Structure occupies higher memory space, union occupies lesser memory space.

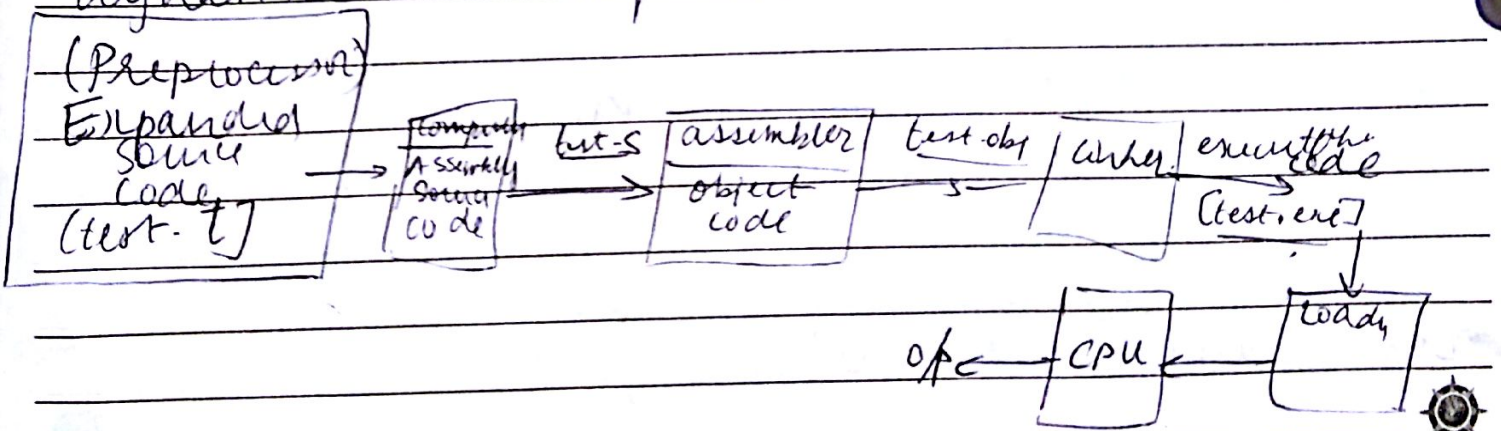
```
strcpy(record2.name, "mut");
printf("%s", record2.name);
strcpy(record2.subject, "Chem");
printf("%d", record2.subject);
```

// Gives correct output.

```
strcpy(record2.name, "mut");
strcpy(record2.subject, "chem");
printf("%s", record2.name);
printf("%s", record2.subject);
```

// correct output for subject, junk value for name.

global initialised → DS (global data stack)
Global uninit → BSS (block stack set)
Local initialised → Stack
Local uninitialised → stack
dynamic → heap



DATE

OS → In charge of memory allocation.

Memory management functions are

(1) calloc ()
(2) Malloc ()
(3) realloc ()
(4) free ()

```
int *ptr;
ptr = malloc (10 * sizeof(int))
```

[Segment uninitialized Junk]

↳ Somewhere in memory 40 bytes get blocked.

heap memory

```
int *ptr;
ptr = calloc (10, 10 * sizeof(int))
```

[Segment uninit. 0]

↳ 10 blocks of 40 bytes each get allocated

Pointer points to that point

realloc () → expand the existing allocated memory.

free () → After usage, the memory has to be deallocated, for this, we use free.

```
ptr = realloc (ptr, 100 * sizeof(int))
```

PES UNIVERSITY

DATE

Memory allocated during at run time is called dynamic memory allocation. Here the size of the memory can be modified during program execution.

Calloc() & malloc()

Malloc() → Only a single block of memory is allocated. No initialisation → garbage value.

calloc() → Multiple blocks of memory is allocated & allocated memories to 0 are.

void *ptr → generic purpose pointer

int inum

ptr = &inum;

struct ex
{
int n1;
int n2;
} s[4];

pass structure to a fn.

void acc(struct ex sptr[], int n)
{
int i;
for(i=0; i<n; i++)
{
printf("Enter num1 \n");
scanf("%d", &s[i].n1);
printf("Enter num2 \n");
}
}

void prog(struct ex sptr[], int n)
{
int i;
for (i=0; i<n; i++)

DATE

```
    {
        printf("Num1: %d\n", sptr[i].n1);
        printf("Num2: %d\n", sptr[i].n2);
    }
}

int main()
{
  ACE(s,3);
  print(s,3);
}
```

individual numbers passed to f^n.

```
display(int *n1, int *n2)
{ printf("%s %s%d\n", );
```

```
ptr = &b1
printf("%s %s %d \n", ptr->name, ptr-> author,
                      ptr-> no);
```

Eg of Function Returning a Structure.

```
struct complexnum (struct comp a, struct comp b)
{
  struct comp c;
  c.real = a.real + b.real;
  c.imag = a.imag + b.imag;
  return c;
}
```

PES UNIVERSITY

DATE

```
struct student *ptr;
ptr = &stud1;
printf("%d", (*ptr).roll);
```

Using pointers

OR.

```
record *ptr;
ptr = (record *) malloc (sizeof(record));
(*ptr.i) = 10;
printf("%d", (*ptr).i);
```

Array of pointers to structure.

Pointer to an array of structures.

Write a program to find the sum of digits in a given long integer.

```
#include <stdio.h>
void main()
{
    long int l;
    printf("Enter a number: ");
    scanf("%ld", &l);
    int dig, sum=0;
    while(l>0)
    {
        dig = l%10;
        l /= 10;
        sum += dig;
    }
    printf("The sum of digits is: %d", sum);
}
```

PES UNIVERSITY

DATE

Write a program to accept a long integer and to extract the most significant digit of the number.

```
#include <stdio.h>
#include <math.h>
void main()
{
  long int l;
  printf("Enter a number: ");
  scanf("%ld", &l);
  int count; long int m; m = l;
  while(l > 0)
  {
    count++;
    l /= 10;
  }
  dig = m / pow(10, count-1);
  printf("%d", m);
}
```

Write a program to reverse a long integer.

```
#include<stdio.h>
#include<math.h>
void main()
{ long n, rev = 0;
  printf("Enter a number: ");
  scanf("%ld", &n);
  while(n)
  {
    rev = rev * 10 + n % 10
    n = n/10;
  }
  printf("%ld", rev);
}
```

PES UNIVERSITY

DATE

Write a program to accept the long integer. Split and store individual digits in an array.

```
#include <stdio.h>
void main( )
{
  long int l;
  printf ("Enter a number: ");
  scanf ("%ld", &l);
  int count; long int m; m = l;
  while (l > 0)
  {
     count ++;
     l /= 10;
  }
  int arr [count]; int i = 0, dig;
  while ( i <= count )
  {  dig = m%10;
     arr [count - i] = dig;
     i++;
     m /= 10;
  }
  for (i=0; i <=count; i++)
     printf("%d", arr[i]);
}
```

Bit fields:

```
struct new
{
  char a: 2,      → no. of bits
  int  b: 4.
  float f: 8;
}
```

bit field can be used only for int & char. If int is an integer of 4 steps of 4 bytes; char only 1 bytes.

PES

```
#define max 20
struct stack
{
   int stack[max]; int top;
};
stack s;
void push(void);
int pop(void)
void display(void);
void main()
{
   int ch;
   int option = 1;
   s.top = -1
   while (option)
   { printf("1 for push, 2 for pop, 3 for display)
     switch (choice)
        case 1:
        :
}
push()
{
   int num;
   if s.top == (max-1)
        printf("stack is full")
   else printf("Enter element)
        s.top = s.top + 1
        s.stack[s.top] = num;
}
```

```
int pop()
{
   if s.top == -1
        empty
        return(s.top);
   else
        num = s.stack[s.top];
        s.top = s.top - 1;
   return(num);

display()
   if s.top == -1
        empty
   else for(i=s.top;i>=
        printf(s.stack[i]
```

DATE

Q. WAP to create a structure student consisting of 2 fields name and USN. Implement an array of structures and sort the elements of this array based on USN

```
Struct student
{
  char name[20];
  int roll;
} stud[60];

Void main()
{  struct student t;
   int n,i,j;
   printf("Enter number of students: ");
   scanf("%d", &n);
   for(i=0; i<n; i++)
      {scanf("%s", &stud[i].name);
       scanf("%d", &stud[i].roll);
      }
   for(i=0; i<n; i++)
      for(j=i; j<n; j++)
         if(stud[j].roll > stud[i].roll)
            t = stud[i];
            stud[j] = stud[i];
            stud[i] = t;
   for(i=0; i<n; i++)
      {
         printf("%s", stud[i].name);
         printf("%d", stud[i].roll);
      }
}
```

DATE

Write A c program to find the largest of 3 numbers without using conditional or relational operators.

```
int a = 1, b = 2, c = 3
int d = a/b
if(d)
   d = a;
else
   d = b;
int e = c/d
if(e)
   d = c
printf("%d", d);
```

Write A program to print numbers from 1 to 100 without using looping constructs.

```
int prn(int n)
{
    if (n==100)
    ;
    else {printf("%d", n)
          prn(++n);
         }
}

void main
{ prn(1);
}
```

```
static int n=0
if (n<100)
  { n++
    printf("%d", n);
    main();
  }
```

DATE

Queue → First in first out

```
#include <stdio.h>
#define max 20
struct queue
{
   int queue[max]; int front, rear;
};
  queue q;
void push()
{
    if q.rear == (max-1)
          // full
    else // input element
          q.rear = q.rear + 1
          q.queue[q.rear] = num
};
  pop
{
   if S.front == -1
         // empty
   return(-1);
   else
       num = q.queue[q.front];
       q.front = q.front + 1
       return(num);
}
```

Node = info + link.

Linked list → Pointer field of the node contains the address of the next block

1st node → 2nd node ..... Last node (Link part is NULL)

DATE

Circular Queue

```
int queue[size], rear = -1, front = -1;

insert()
{ if ((front == 0 && rear == size-1) || (front == rear+1))
  printf("\n\n queue is full");
else
 {
  printf("\n\n enter item: ");
  scanf("%d", &item);
  if (rear == -1)
   {
     rear = 0;
     front = 0;
   }
else if (rear == size-1)
    rear = 0;
else
    rear = rear+1
    queue[rear] = item;
}

delete()
{ if ((front == -1) || front == rear+1))
  printf("\n\n empty");
  else
      item = queue[front];
      if (front == rear)
        {
          front = -1;
          rear = -1;
        }
else if (front == size-1)
    front = 0;
```

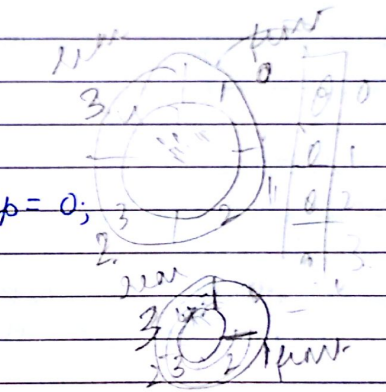
DATE

```
else
    front ++;
}

display()
if((front == -1))
{ //empty
}
else if (rear == front)
{
  for(i=front; i<=size-1; i++
      printf()
  for(j=0; j<=rear; j++)
      printf()
```

Linked list-

```
struct node
  { int num;
    struct node *ptr;
  };
node *head, *first, *temp=0;
int count=0;
int choice=1
first=0
while(choice)
{
  head=(node*)malloc(sizeof(node));
  printf("Enter data item\n");
  scanf("%d", &head->num);
  if(first!=0)
  {
    temp->ptr=head;
    temp=head;
```



DATE

```
else
    front ++;
}

display()
if((front == -1))
{ //empty
}
else if (rear == front)
{
  for (i=front; i<=size-1; i++
      printf()
  for (j=0; j<=rear; j++)
      printf()
```

Linked list.

```
struct node
   { int num;
     struct node *ptr;
   };
node *head, *first, *temp=0;
int count=0;
int choice=1
first=0
while(choice)
{
   head=(node*)malloc(sizeof(node));
   printf("Enter data item\n");
   scanf("%d", &head->num);
   if(first!=0)
   {
     temp->ptr=head;
   } temp=head;
```

DATE

```
        else
            first = temp = head;
    }
    //do you want to continue
    //scanf
    temp -> ptr = 0
    temp = first;

    while(temp != 0)
    {
        printf("%d ->", temp -> num);
        count++;
        temp = temp -> ptr;
    }
    printf("NULL");
```

Stack using linked list:

```
struct node
{
    int data;
    struct node * ptr;
};
struct node * top = NULL;
void init (struct node * head)
{
    head = NULL;
}

struct node * push (struct node * head, int data)
{
    struct node * temp = (struct node *) malloc (sizeof(struct
    if(temp == NULL)
        { exit(0); }
```

DATE

```
temp -> data = data;
temp -> next = top;
top = temp
return next;
}
```

```
if(top == NULL)
{ top = temp;
  top -> next = NULL;
}
```

```
struct node * pop(struct node * head)
{ if(top == NULL) -> underflow
  else
  { t = top;
    info = top -> data;
    top = top -> link
    t -> link = NULL
    free(t)
    return(info)
}
```

```
display()
{ struct node *t
  if(temp == NULL)
      //empty
  else
    t = top;
  while(t != NULL)
  {
    printf("%d", t -> data);
    t = t -> link
  }
}
```

Linked queue.

DATE

```
struct node
{
  int data;
  struct node *next;
} *front, *rear;

void delete ()
{
  struct node *tmp, *var = front;
  if(var == front)
  {
    front = front -> next;
    free(var);
  }
else
    printf("Empty");
}

void push(int value)
{
  struct node *tmp;
  tmp = (struct node*)malloc(sizeof(struct node));
  temp -> data = value
  if(rear == NULL
  {
    rear = tmp;
    rear -> next = NULL;
  }
else
  { tmp -> next = rear;
  } rear = tmp
}
```

```
    }
    while (var != NULL)
    {
        printf("%d", var->data);
        var = var->next;
    }
}
else printf("Empty \n");
}
```