STRUCTURES and POINTERS

```
#include <stdio.h>
struct date{
   int day;
   int month;
   int year;
};
int main()
{
   struct date *datePtr;;
   struct date todaysDate;
   datePtr=&todaysDate;
   /*
   (*datePtr).day=22;
   (*datePtr).month=11;
   (*datePtr).year=2024;
   */
   datePtr->day=22;
   datePtr->month=11;
   datePtr->year=2024;
   printf("todays date : %d-%d-%d\n",(*datePtr).day,(*datePtr).month,(*datePtr).year);//brackets are mandatory
bcoz '.' operator has higher precedence than '*'.but we need derefrencing fist(i.e '*')
   printf("using -> operator\n");
   printf("todays date : %d-%d-%d\n",datePtr->day,datePtr->month,datePtr->year);//-> operator
   return 0;
}
```

STRUCTURES CONTAINING POINTERS

```
#include <stdio.h>
struct pnum{
   int *p1;
   int *p2;
};
int main(){
   int i1=20,i2;
   struct pnum pointers;
   pointers.p2=&i2;
   pointers.p1=&i1;
   *pointers.p2=80;
   printf("i1=%d\n",*pointers.p1);
   printf("i2=%d",*pointers.p2);
}
```

```
#include <stdio.h>

struct names{
   char first[40];
   char last[40];
};

struct pNames{
   char *first;
   char *last;
};
```

```
int main(){
   struct names CAnames ={"Abhinav", "Karan"};
   struct pNames CPnames = {"Abhinav","Karan"};

   printf("%s\t%s \n",CAnames.first,CPnames.first);
   printf("size of CAnames = %d\n",sizeof(CAnames));
   printf("size of CPnames = %d\n",sizeof(CPnames));
   return 0;
}
```

STRUCTURE AS ARGUMENTS IN FUNCTION

Call by value

```
#include <stdio.h>
#include<string.h>
#include<stdbool.h>
struct names{
   char first[40];
   char last[40];
};

struct pNames{
   char *first;
   char *last;
};
 bool nameCompare(struct names, struct names);
int main(){
   struct names CAnames ={"Abhinav", "Karan"};
   struct names CPnames = {"Abhinav","Karan"};
   bool b=nameCompare(CAnames,CPnames);
   printf("%d",b);
   return 0;
}
bool nameCompare(struct names CAnames,struct names CPnames){
   if(strcmp(CAnames.first,CPnames.first)==0){
      return true;
   }
   else{
      return false;
   }
}
```

Call by reference

```
#include <stdio.h>
#include<string.h>
#include<stdbool.h>
struct names{
   char first[40];
   char last[40];
};


 bool nameCompare(struct names *, struct names *);
int main(){
   struct names CAnames ={"Abhinav", "Karan"};
   struct names CPnames = {"Abhinav","Karan"};
```

```
  struct names *ptr1,*ptr2;
  ptr1=&CAnames;
  ptr2=&CPnames;
  bool b=nameCompare(ptr1,ptr2);
  //bool b=nameCompare(&CAnames,&CPnames);//giving address directly,c
  printf("%d",b);
  return 0;
}
bool nameCompare(struct names *ptr1,struct names *ptr2){
  if(strcmp(ptr1->first,ptr2->first)==0){
    return true;
  }
  else{
    return false;
  }
}
```

Problem 1: Dynamic Student Record Management

Objective: Manage student records using pointers to structures and dynamically allocate memory for student names.

Description:

1. Define a structure Student with fields:
   - int roll_no: Roll number
   - char *name: Pointer to dynamically allocated memory for the student's name
   - float marks: Marks obtained
2. Write a program to:
   - Dynamically allocate memory for n students.
   - Accept details of each student, dynamically allocating memory for their names.
   - Display all student details.
   - Free all allocated memory before exiting.
   -

```
#include <stdio.h>
#include <string.h>
#include<stdlib.h>
struct student {
  char *name;
  int rollNum;
  float marks;
};
void addStudent(struct student *,int);
void printStudentData(struct student*,int);

int main(){
  int no,option;

  struct student *p_stdDetails;
  int flag = 0;
  while(1){
    printf("Enter your choice\n1-add data\n2-display all data\n3-exit:\n");
    scanf("%d", &option);
    switch(option){
      case 1:

        printf("enter number of student details to be entered:");
        scanf("%d",&no);
        p_stdDetails=(struct student*)malloc(no*sizeof(struct student));
```

```
            addStudent(p_stdDetails,no);
            flag=1;
            break;
          case 2:
            if(flag==0){
              printf("no data enterd.\n");
              return 0;
            }
            printStudentData(p_stdDetails,no);
            break;
          case 3:
            free(p_stdDetails);
            free(p_stdDetails->name);
            printf("exiting!!");
            return 0;
          default:
            printf("invalid option");
        }

    }
}
void addStudent(struct student *p_stdDetails,int no) {
    for(int i=0;i<no;i++){
      p_stdDetails[i].name=(char*)malloc(10*sizeof(char));
      printf("Name: ");
       scanf(" %[^\n]", p_stdDetails[i].name);

      printf("Roll Number: ");
      scanf("%d", &p_stdDetails[i].rollNum);

      printf("Marks: ");
      scanf("%f", &p_stdDetails[i].marks);
    }
}
void printStudentData(struct student *p_stdDetails,int no){
    for(int i=0;i<no;i++){
      printf("%s || %d || %f\n",p_stdDetails[i].name,p_stdDetails[i].rollNum,p_stdDetails[i].marks);
    }
}
```

Problem 2: Library System with Dynamic Allocation

Objective: Manage a library system where book details are dynamically stored using pointers inside a structure.

Description:

1. Define a structure Book with fields:
   - char *title: Pointer to dynamically allocated memory for the book's title
   - char *author: Pointer to dynamically allocated memory for the author's name
   - int *copies: Pointer to the number of available copies (stored dynamically)
2. Write a program to:
   - Dynamically allocate memory for n books.
   - Accept and display book details.
   - Update the number of copies of a specific book.
   - Free all allocated memory before exiting.

```
#include <stdio.h>
#include <string.h>
#include<stdlib.h>
```

```
struct library{
  char *title;
  char *author;
  int  *copies;
};
void acceptAndDisplay(struct library*,int);
int updateCopies(struct library*,char*,int);

int main(){
  int no,option;
  struct library *books;
  int flag = 0;
  while(1){
    printf("Enter your choice\n1-add and dispaly data\n2-update copies\n3-exit:\n");
    scanf("%d", &option);
    switch(option){
      case 1:
        printf("enter number of books details to be entered:");
        scanf("%d",&no);
        books=(struct library*)malloc(no*sizeof(struct library));
        acceptAndDisplay(books,no);
        flag=1;
        break;
      case 2:
        if(flag==0){
          printf("no data enterd.\n");
          return 0;
        }
        printf("enter title of the book\n");
        char nTitle[20];
        scanf(" %[^\n]",nTitle);
        updateCopies(books,nTitle,no);
        printf("updated no of copies\n");

        break;
      case 3:
        free(books);
        free(books->title);
        free(books->author);
        free(books->copies);
        printf("exiting!!");
        return 0;
      default:
        printf("invalid option");
    }
  }
}
void acceptAndDisplay(struct library *books,int no){
  printf("enter %d books\n",no);

  for(int i=0;i<no;i++){
    books[i].title=(char*)malloc(20*sizeof(char));
    books[i].author=(char*)malloc(20*sizeof(char));
    books[i].copies=(int*)malloc(1*sizeof(int));
    printf("title: ");
    scanf(" %[^\n]", books[i].title);
```

```
    printf("author: ");
    scanf(" %[^\n]", books[i].author);

    printf("copies: ");
    scanf("%d", books[i].copies);
  }
  printf("all available books\ntitle|\t\tauthor|\t\tcopies|\t\t\n");
  for(int i=0;i<no;i++){
    printf("%s\t\t%s\t\t %d\t\t\n", books[i].title,books[i].author,*(books[i].copies));
  }
}
int updateCopies(struct library *books,char *nTitle,int no){
  int i=0;
  while(i<no){
    if(strcmp(nTitle,books[i].title)==0){
      printf("book found\n");
      printf("enter new no of copies:");
      scanf("%d",books[i].copies);
      return i;
    }
    i++;
  }
  return -1;
}
```

Problem 1: Complex Number Operations

Objective: Perform addition and multiplication of two complex numbers using structures passed to functions.

Description:

1. Define a structure Complex with fields:
   - float real: Real part of the complex number
   - float imag: Imaginary part of the complex number
2. Write functions to:
   - Add two complex numbers and return the result.
   - Multiply two complex numbers and return the result.
3. Pass the structures as arguments to these functions and display the results.

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
struct complex{
  float real;
  float imag;
};
void addComplex(struct complex*);
void mulComplex(struct complex*);
int main(){
  struct complex *p_complex;
  printf("enter two complex numbers\n");
  p_complex=(struct complex*)malloc(2*sizeof(struct complex));
  for(int i=0;i<2;i++){
    printf("enter real part of %d number: ",i+1);
    scanf("%f",&p_complex[i].real);
    printf("enter imag part of %d number: ",i+1);
    scanf("%f",&p_complex[i].imag);
  }
  printf("2 complex numbers are \n");
```

```
  for(int i=0;i<2;i++){
    printf("%.2f+%.2fj\n",p_complex[i].real,p_complex[i].imag);
  }
  addComplex(p_complex);
  mulComplex(p_complex);
  return 0;
}
void addComplex(struct complex *p_complex){
  float sum_r=0,sum_i=0;

    sum_r=p_complex[0].real+p_complex[1].real;
    sum_i=p_complex[0].imag+p_complex[1].imag;

  printf("sum is %.2f+%.2fj\n",sum_r,sum_i);
}
void mulComplex(struct complex *p_complex){
  float pdt_r=0,pdt_i=0;
  pdt_r=(p_complex[0].real*p_complex[1].real)-(p_complex[0].imag*p_complex[1].imag);
  pdt_i=(p_complex[0].real*p_complex[1].imag)+(p_complex[0].imag*p_complex[1].real);
  printf("pdt is %.2f+%.2fj\n",pdt_r,pdt_i);

}
```

Problem 2: Rectangle Area and Perimeter Calculator

Objective: Calculate the area and perimeter of a rectangle by passing a structure to functions.

Description:

1. Define a structure Rectangle with fields:
    - float length: Length of the rectangle
    - float width: Width of the rectangle
2. Write functions to:
    - Calculate and return the area of the rectangle.
    - Calculate and return the perimeter of the rectangle.
3. Pass the structure to these functions by value and display the results in main.

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
struct rectangle{
  float length;
  float width;
};
float area(struct rectangle);
float perimeter(struct rectangle rect_1);
int main(){
  struct rectangle rect_1={20,10};
  float rectArea=area(rect_1);
  printf("area of rectangle is %.2f\n",rectArea);
  float rectPeri=perimeter(rect_1);
  printf("perimeter of rectangle is %.2f\n",rectPeri);
  return 0;
}
float area(struct rectangle rect_1){
  float rectArea=0;
  rectArea=rect_1.length*rect_1.width;
  return rectArea;

}
```

```
float perimeter(struct rectangle rect_1){
  float rectPeri=0;
  rectPeri=(rect_1.length+rect_1.width)*2;
  return rectPeri;
}
```

Problem 3: Studentstru Grade Calculation

Objective: Calculate and assign grades to students based on their marks by passing a structure to a function.

Description:

1. Define a structure Student with fields:
   - char name[50]: Name of the student
   - int roll_no: Roll number
   - float marks[5]: Marks in 5 subjects
   - char grade: Grade assigned to the student
2. Write a function to:
   - Calculate the average marks and assign a grade (A, B, etc.) based on predefined criteria.
3. Pass the structure by reference to the function and modify the grade field.

```
 #include<stdio.h>
#include<stdlib.h>
#include<string.h>
struct student{
  char name[50];
  int roll ;
  float mark[5];
  char grade;
};
char grade(struct student*,int);
int main(){
  int no;
  printf("enter number of students details to be entered");
  scanf("%d",&no);
  struct student *ptrStudents;
  ptrStudents=(struct student *)malloc(no*sizeof(struct student));
  printf("enter the student details\n");
  for(int i=0;i<no;i++){
    printf("enter %d student details\n",i+1);
    printf("name:");
    scanf(" %[^\n]",ptrStudents[i].name);
    printf("roll no:");
    scanf("%d",&ptrStudents[i].roll);
    for(int j=0;j<5;j++){
      printf("mark out of 100\n");
      printf("mark%d:",j+1);
      float MARK;
      scanf("%f",&MARK);
      if(MARK>100){
        printf("max mark is 100");
        return 0;
      }
      else{
        ptrStudents[i].mark[j]=MARK;
      }
    }
    char GRADE;
    GRADE=grade(ptrStudents,i);
```

```
    ptrStudents[i].grade=GRADE;
  }
  printf("\nStudent Details:\n");
  printf("Name\t\tRoll No\t\tMarks\t\t\t\tGrade\n");
  printf("*****************************************************************\n");
  for (int i = 0; i < no; i++) {
    printf("%s\t\t%d\t\t", ptrStudents[i].name, ptrStudents[i].roll);
    for (int j = 0; j < 5; j++) {
      printf("%.2f ", ptrStudents[i].mark[j]);
    }
    printf("\t\t%c\n", ptrStudents[i].grade);
  }

  return 0;
}
char grade(struct student *ptrStudents,int count){
  float sum=0,avg;
  for(int i=0;i<5;i++){
    sum=sum+ptrStudents[count].mark[i];
  }
  avg=sum/5;
  if(avg>=90&&avg<=100){
    return 'A';
  }
  else if(avg>=80&&avg<89){
    return 'B';
  }
  else if(avg>=70&&avg<79){
    return 'C';
  }
  else if(avg>=60&&avg<69){
    return 'D';
  }
  else if(avg<60){
    return 'F';
  }
}
```

Problem 4: Point Operations in 2D Space

Objective: Calculate the distance between two points and check if a point lies within a circle using structures.

Description:

1. Define a structure Point with fields:
   - float x: X-coordinate of the point
   - float y: Y-coordinate of the point
2. Write functions to:
   - Calculate the distance between two points.
   - Check if a given point lies inside a circle of a specified radius (center at origin).
3. Pass the Point structure to these functions and display the results.

```
#include <stdio.h>

struct Point {
  float x;
  float y;
};
```

```
int inCircle(struct Point, float);
int main() {
  struct Point p;
  float radius;
  printf("Enter the coordinates of the point (x y): ");
  scanf("%f %f", &p.x, &p.y);
  printf("Enter the radius of the circle (centered at origin): ");
  scanf("%f", &radius);
  if (inCircle(p, radius)) {
    printf("The point (%.2f, %.2f) lies inside the circle.\n", p.x, p.y);
  } else {
    printf("The point (%.2f, %.2f) lies outside the circle.\n", p.x, p.y);
  }

  return 0;
}


int inCircle(struct Point p, float radius) {
  float Square = p.x * p.x + p.y * p.y;


  if (Square <= radius * radius) {
    return 1;
  } else {
    return 0;
  }
}
```

Problem 5: Employee Tax Calculation

Objective: Calculate income tax for an employee based on their salary by passing a structure to a function.

Description:

1. Define a structure Employee with fields:
   - char name[50]: Employee name
   - int emp_id: Employee ID
   - float salary: Employee salary
   - float tax: Tax to be calculated (initialized to 0)
2. Write a function to:
   - Calculate tax based on salary slabs (e.g., 10% for salaries below $50,000, 20% otherwise).
   - Modify the tax field of the structure.
3. Pass the structure by reference to the function and display the updated tax in main.

```
#include <stdio.h>
#include <string.h>

struct Employee {
  char name[50];
  int emp_id;
  float salary;
  float tax;
};
void calculateTax(struct Employee*);
int main() {
  struct Employee emp;
  printf("Enter employee details:\n");
  printf("Name: ");
  scanf(" %[^\n]", emp.name);
```

```
  printf("Employee ID: ");
  scanf("%d", &emp.emp_id);
  printf("Salary: ");
  scanf("%f", &emp.salary);
  calculateTax(&emp);
  printf("\nEmployee Details:\n");
  printf("Name: %s\n", emp.name);
  printf("Employee ID: %d\n", emp.emp_id);
  printf("Salary: $%.2f\n", emp.salary);
  printf("Calculated Tax: $%.2f\n", emp.tax);

  return 0;
}
void calculateTax(struct Employee* emp) {
  if (emp->salary < 50000) {
    emp->tax = emp->salary * 0.10;
  } else {
    emp->tax = emp->salary * 0.20;
  }
}
```

Problem Statement: Vehicle Service Center Management

Objective: Build a system to manage vehicle servicing records using nested structures.

Description:

1. Define a structure Vehicle with fields:
   - char license_plate[15]: Vehicle's license plate number
   - char owner_name[50]: Owner's name
   - char vehicle_type[20]: Type of vehicle (e.g., car, bike)
2. Define a nested structure Service inside Vehicle with fields:
   - char service_type[30]: Type of service performed
   - float cost: Cost of the service
   - char service_date[12]: Date of service
3. Implement the following features:
   - Add a vehicle to the service center record.
   - Update the service history for a vehicle.
   - Display the service details of a specific vehicle.
   - Generate and display a summary report of all vehicles serviced, including total revenue.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

struct Vehicle {
  char license[15];
  char owner[50];
  char vehicle_type[20];
};

struct Service {
  struct Vehicle VehicleService;
  char service_type[30];
  float cost;
  char date[12];
};


int ServiceData(struct Service *, int, struct Vehicle *);
```

```
int VehicleData(struct Vehicle *, int);
void displayServiceData(struct Service *, char *, int);
void generateReport(struct Service *, int);
void updateServiceDate(struct Service *, int, char *, char *);

int main() {

  struct Vehicle *vehicleRecord;
  struct Service *serviceRecord;
  int no_v = 0, no_s = 0;


  vehicleRecord = (struct Vehicle*)malloc(10 * sizeof(struct Vehicle));
  serviceRecord = (struct Service*)malloc(10 * sizeof(struct Service));
  while(1){
    printf("----------------------------------------------------------------\n");
    printf("1-add Vehicle to data\n2-update service history\n3-search Vehicle derails by license\n4-display
all serviced vehicles and total revenue\n");
    printf("----------------------------------------------------------------\n");
    int op;
    printf("enter operation:");
    scanf("%d",&op);
    switch(op){
      case 1:
        no_v = VehicleData(vehicleRecord, no_v);
        no_s = ServiceData(serviceRecord, no_s, vehicleRecord);
        break;
      case 2:
        char licenseToSrch[15];
        printf("Enter license to search: ");
        scanf(" %[^\n]", licenseToSrch);
        printf("--------------------------------\n");
        displayServiceData(serviceRecord, licenseToSrch, no_s);
        break;
      case 3:

        printf("Enter license to search: ");
        scanf(" %[^\n]", licenseToSrch);
        char newDate[12];
        printf("Enter new service date (dd-mm-yy): ");
        scanf(" %[^\n]", newDate);
        updateServiceDate(serviceRecord, no_s, licenseToSrch, newDate);
        break;
      case 4:
        printf("\nSummary Report:\n");
        printf("--------------------------------\n");
        generateReport(serviceRecord, no_s);
        break;

      default:
        free(vehicleRecord);
        free(serviceRecord);
        printf("invalid operation\n");
        printf("exiting system!!!!!");
        return 0;
    }
```

```
    }
}

// Function to add vehicle data
int VehicleData(struct Vehicle *vehicleRecord, int index) {
    printf("Enter Vehicle details\n");
    printf("--------------------------------\n");
    printf("License: ");
    scanf(" %[^\n]", vehicleRecord[index].license);
    printf("Owner Name: ");
    scanf(" %[^\n]", vehicleRecord[index].owner);
    printf("Vehicle Type: ");
    scanf(" %[^\n]", vehicleRecord[index].vehicle_type);
    printf("--------------------------------\n");
    index++;
    return index;
}

// Function to add service data
int ServiceData(struct Service *serviceRecord, int index, struct Vehicle *vehicleRecord) {
    printf("Details of vehicle entered for service\n");
    printf("--------------------------------\n");
    serviceRecord[index].VehicleService = vehicleRecord[index];
    printf("Service Type: ");
    scanf(" %[^\n]", serviceRecord[index].service_type);
    printf("Cost: ");
    scanf(" %f", &serviceRecord[index].cost);
    printf("Date (dd-mm-yy): ");
    scanf(" %[^\n]", serviceRecord[index].date);
    printf("--------------------------------\n");
    index++;
    return index;
}

// Function to display service details of a specific vehicle
void displayServiceData(struct Service *serviceRecord, char *licenseToSrch, int no_s) {
    int found = 0;
    for (int i = 0; i < no_s; i++) {
        if (strcmp(serviceRecord[i].VehicleService.license, licenseToSrch) == 0) {
            printf("Vehicle Found\n");
            printf("License No: %s\n", serviceRecord[i].VehicleService.license);
            printf("Owner: %s\n", serviceRecord[i].VehicleService.owner);
            printf("Vehicle Type: %s\n", serviceRecord[i].VehicleService.vehicle_type);
            printf("Service Type: %s\n", serviceRecord[i].service_type);
            printf("Cost: %.2f\n", serviceRecord[i].cost);
            printf("Service Date: %s\n", serviceRecord[i].date);
            printf("--------------------------------\n");
            found = 1;
            break;
        }
    }
    if (found == 0) {
        printf("Vehicle not found.\n");
    }
}
```

```
// Function to generate and display a summary report of all vehicles serviced
void generateReport(struct Service *serviceRecord, int no_s) {
  float totalRevenue = 0;
  for (int i = 0; i < no_s; i++) {
    printf("License No: %s\n", serviceRecord[i].VehicleService.license);
    printf("Owner: %s\n", serviceRecord[i].VehicleService.owner);
    printf("Vehicle Type: %s\n", serviceRecord[i].VehicleService.vehicle_type);
    printf("Service Type: %s\n", serviceRecord[i].service_type);
    printf("Cost: %.2f\n", serviceRecord[i].cost);
    printf("Service Date: %s\n", serviceRecord[i].date);
    printf("--------------------------------\n");

    // Add the service cost to total revenue
    totalRevenue += serviceRecord[i].cost;
  }
  printf("\nTotal Revenue: %.2f\n", totalRevenue);
}

// Function to update the service date for a specific vehicle by license plate
void updateServiceDate(struct Service *serviceRecord, int no_s, char *licenseToSrch, char *newDate) {
  int found = 0;
  for (int i = 0; i < no_s; i++) {
    if (strcmp(serviceRecord[i].VehicleService.license, licenseToSrch) == 0) {
      printf("Updating service date for vehicle with license %s...\n", licenseToSrch);
      strcpy(serviceRecord[i].date, newDate);
      printf("Service date updated to: %s\n", serviceRecord[i].date);
      found = 1;
      break;
    }
  }
  if (found == 0) {
    printf("Vehicle not found. Service date not updated.\n");
  }
}
```