USN

Solution with scheme-Model Answer

Internal Assessment Test 3 – June 2024

Sub	**Principles of Programming Using C**					Sub code	**BPOPS203**	Branch	**CSE, CSE(DS)**
Date	**26.06.2024**	Duration	**90 mins**	Max Marks	**50**	Sem /Sec	**II Sem P-Cycle (I,J,K,L)**		**OBE**

Answer any FIVE FULL Questions		MARKS	CO	RBT
1.	**Implement structures to read, write and compute average of marks and the students scoring above and below average marks for class of N students.**	**[10]**	**CO5**	**L3**

- **Algorithm/Flowchart (1)**
- **Program (8)**
- **Output (1)**

Program

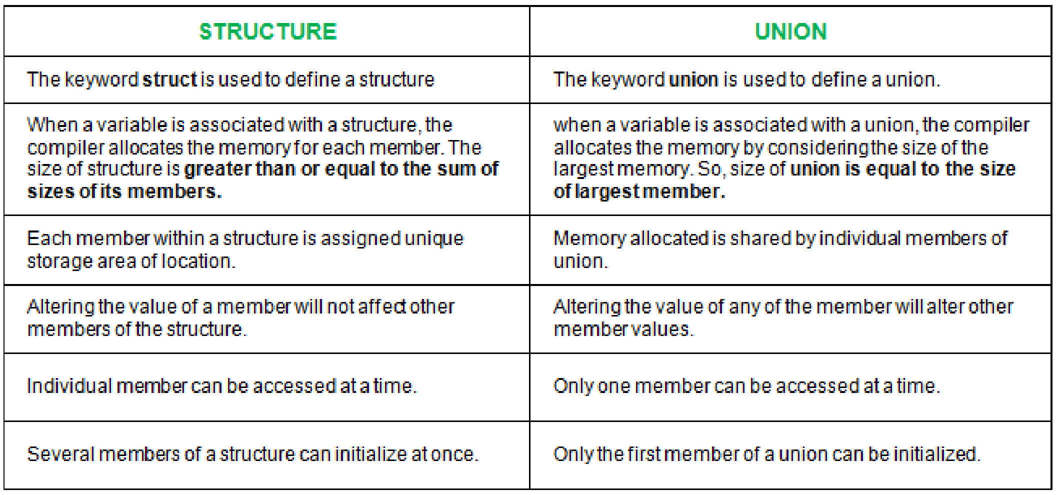
```
#include<stdio.h>
struct student
        {
                int id;
                char name[20];
                float sub[6];
                float avg;
        };
int main()
{
        struct student s[20];
        float sum=0;
        int i,j,n;

        // Accept the number of records/students
        printf("Enter the number of records :");
        scanf("%d",&n);

        // Accept data for all the fields/members of each record
        printf("Enter %d student details...\n",n);

        for(i=0;i<n;i++)
        {
                printf("\n\nEnter student ID, name :");          // Student ID
                scanf("%d%s",&s[i].id, s[i].name);
                printf("Enter 6 subject marks :");

                for (j=0;j<6;j++)
                        {
                                scanf("%f", &s[i].sub[j]);
```

```
                    }
        }

        // Compute the average of each student

        for(i=0;i<n;i++)
        {
                sum=0;
                for (j=0;j<6;j++)
                        {
                                sum = sum + s[i].sub[j];
                        }
                s[i].avg = sum / 6;
        }

        // Display student ID, name and average of all students
        // who have scored above average marks
        printf("Students scoring above the average marks....\n");
        printf("\n\nID\tName\tAverage\n\n");

        for(i=0;i<n;i++)
        {
                if(s[i].avg>=35.0)
                printf("%d\t%s\t%f\n",s[i].id,s[i].name,s[i].avg);
        }

        // Display student ID, name and average of all students
        // who have scored below average marks

        printf("\n\nStudents scoring below the average marks....\n");
        printf("\n\nID\tName\tAverage\n\n");

        for(i=0;i<n;i++)
        {
                if(s[i].avg<35.0)
                printf("%d\t%s\t%f\n",s[i].id,s[i].name,s[i].avg);
        }

        return 0;
}
```

2. a.

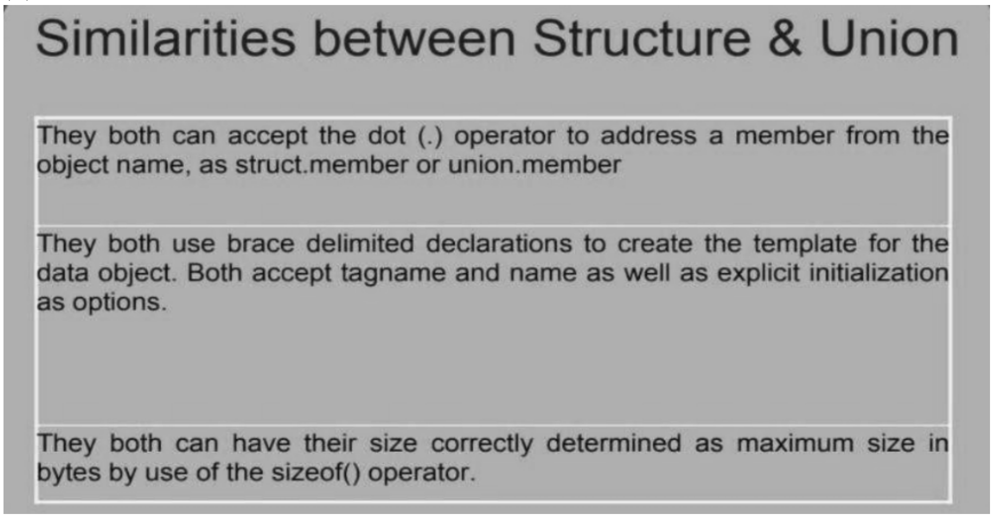
Differentiate Structure and Union.

- **five points (5x1=5)**



STRUCTURE	UNION
The keyword **struct** is used to define a structure	The keyword **union** is used to define a union.
When a variable is associated with a structure, the compiler allocates the memory for each member. The size of structure is **greater than or equal to the sum of sizes of its members.**	when a variable is associated with a union, the compiler allocates the memory by considering the size of the largest memory. So, size of **union is equal to the size of largest member.**
Each member within a structure is assigned unique storage area of location.	Memory allocated is shared by individual members of union.
Altering the value of a member will not affect other members of the structure.	Altering the value of any of the member will alter other member values.
Individual member can be accessed at a time.	Only one member can be accessed at a time.
Several members of a structure can initialize at once.	Only the first member of a union can be initialized.

Write the similarities between Structure and Union. (5)



Similarities between Structure & Union

They both can accept the dot (.) operator to address a member from the object name, as struct.member or union.member

They both use brace delimited declarations to create the template for the data object. Both accept tagname and name as well as explicit initialization as options.

They both can have their size correctly determined as maximum size in bytes by use of the sizeof() operator.

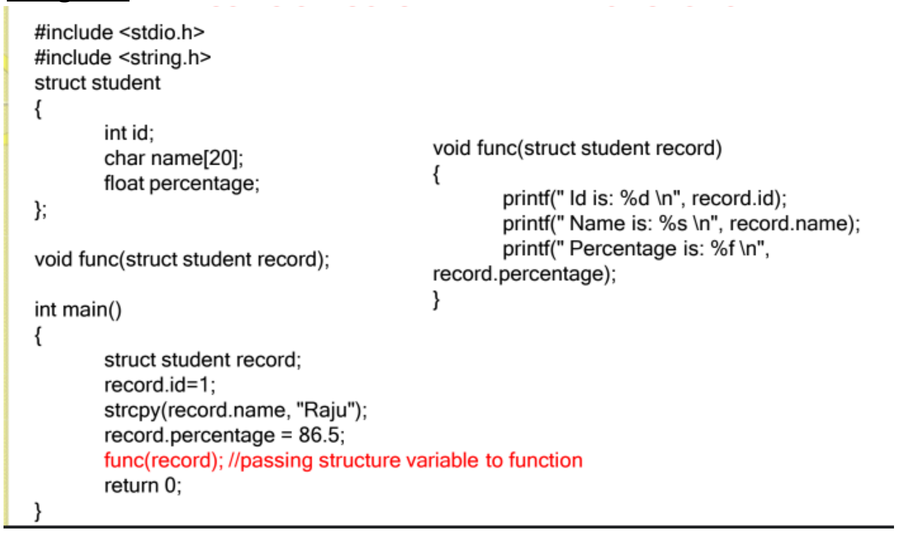
[5] **CO5** **L2**

3. **Write a C program that accepts a structure variable as a parameter to a function from a function call.**

- **algorithm/flowchart (1)**
- **program (8)**
- **output(1)**

[10] **CO5** **L3**

Program



```
#include <stdio.h>
#include <string.h>
struct student
{
        int id;
        char name[20];
        float percentage;
};

void func(struct student record);

int main()
{
        struct student record;
        record.id=1;
        strcpy(record.name, "Raju");
        record.percentage = 86.5;
        func(record); //passing structure variable to function
        return 0;
}

void func(struct student record)
{
        printf(" Id is: %d \n", record.id);
        printf(" Name is: %s \n", record.name);
        printf(" Percentage is: %f \n",
record.percentage);
}
```

4. **What are enumeration variable? How are they declared? Write a C program to demonstrate the working of enum.**

- **enum variable (2)**
- **declaration of enum (2)**
- **program (6)**

[10] **CO5** **L3**

An **enumeration** is a user-defined data type that consists of integral constants. To define an enumeration, keyword enum is used.
Syntax:
enum enum_name{int_const1, int_const2, int_const3, ... int_constN};

Declaration of enumerated variable
enum Boolean {false,true};
enum boolean check;
Here, a variable check is declared which is of type enum boolean.

Program
```
#include <stdio.h>
enum week{ sunday, monday, tuesday, wednesday, thursday,
friday,saturday};
int main()
{
enum week today;
today=wednesday;
printf("%d day",today+1);
return 0;
```

	} Output 4 day			

5	**Write a C program defining a union 'Job' that can store a float 'salary' and an int 'worker number'. Also assign values to these members and print them.**
	• **algorithm/flowchart (1)**
	• **program (8)**
	• **output (1)**
	
	Program
	`#include <stdio.h>`
	
	`// Define a union named Job`
	`union Job {`
	`    float salary;`
	`    int workerNumber;`
	`};`
	
	`int main() {`
	`    // Declare a variable of type Job`
	`    union Job job;`
	
	`    // Assign a value to salary member`
	`    job.salary = 2500.75;`
	
	`    // Print the value of salary`
	`    printf("Salary: %.2f\n", job.salary);`
	
	`    // Now assign a value to worker number member`
	`    job.workerNumber = 12345;`
	
	`    // Print the value of worker number`
	`    printf("Worker Number: %d\n", job.workerNumber);`
	
	`    return 0;`
	`}`
		[10]	**CO5**	**L3**

6	**Write a short note on functions used to** **i) Read data from a file with example** Functions Used for Reading Data from a File: (5) **1.** fopen: Opens a file and returns a pointer to `FILE` structure. 2. fscanf: Reads formatted data from a file. 3. fclose: Closes the file **ii) Write data to a file with example** **Functions Used for Writing Data to a File: (5)** **1.** fopen: Opens a file and returns a pointer to `FILE` structure.	**[5]** **[5]**	**CO5** **CO5**	**L2** **L2**

2. fprintf: Writes formatted data to a file.
3. fclose: Closes the file.

—---

File handling in C involves using functions from the standard I/O library (`stdio.h`). Here's a brief overview of the key functions used for file handling along with examples**:**

Functions for File Handling in C:

1. **fopen**: Opens a file and returns a pointer to a `FILE` structure.

syn: FILE *fopen(const char *filename, const char *mode);

- ****Example****:

```
FILE *fp;

fp = fopen("file.txt", "r"); // Opens file.txt in read mode

if (fp == NULL) {

    perror("Error opening file");

    return 1;

}
```

2. **fclose**: Closes a file.

syn: int fclose(FILE *stream);

****Example****:

```
fclose(fp); // Closes the file pointed by fp
```

3. **fgetc** and **fputc**: Read and write a single character from/to a file.

syn: int fgetc(FILE *stream);

syn: int fputc(int character, FILE *stream);

****Example**** (reading characters from a file and printing them):

```
  int c;

while ((c = fgetc(fp)) != EOF) {

    putchar(c); // Output character to console
}
```

4. **fgets** and **fputs**: Read and write a string from/to a file.

syn: char *fgets(char *str, int n, FILE *stream);

syn: int fputs(const char *str, FILE *stream);

****Example**** (reading lines from a file and printing them):

```
char buffer[100];
while (fgets(buffer, sizeof(buffer), fp) != NULL) {
    printf("%s", buffer); // Output line to console
}
```

5. **fread** and **fwrite**: Read and write blocks of data from/to a file.

syn: size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);

syn: size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream);

****Example**** (copying contents from one file to another):

```
char buffer[1024];
size_t bytesRead;
while ((bytesRead = fread(buffer, 1, sizeof(buffer), sourceFile)) > 0) {
    fwrite(buffer, 1, bytesRead, targetFile);
}
```

6. **fprintf** and **fscanf**: Read and write formatted data from/to a file.

syn: int fprintf(FILE *stream, const char *format, ...);

syn: int fscanf(FILE *stream, const char *format, ...);

****Example**** (writing formatted data to a file):

```
fprintf(fp, "Name: %s, Age: %d\n", "John Doe", 25);
```

****Example**** (reading formatted data from a file):

```
char name[50];
```

	`int age;`
	
	`fscanf(fp, "Name: %s, Age: %d\n", name, &age);`			

7	**Write a C program to copy a text file to another, read both the input file name and target file name.**
	
	● **algorithm/flowchart (1)**
	● **program (8)**
	● **output (1)**
	
	<u>Program</u>
	`#include<stdio.h>`
	`int main()`
	`{`
	`    char src_fname[20], tar_fname[20], ch;`
	`    printf("Enter input file name and target file name :");`
	`    scanf("%s%s", src_fname, tar_fname);`
	`    FILE *fp1,*fp2;`
	`    fp1 = fopen(src_fname, "r");`
	`    if (fp1 == NULL)`
	`    {`
	`        printf("Unable to open file - %s in Read mode\n",src_fname);`
	`        return 1;`
	`    }`
	`    fp2 = fopen(tar_fname, "w");`
	`    if (fp2 == NULL)`
	`    {`
	`        printf("Unable to open file - %s in write mode\n", tar_fname);`
	`        return 2;`
	`    }`
	`    while ((ch = fgetc(fp1)) != EOF)`
	`    {`
	`            fputc(ch, fp2);`
	`    }`
	`    printf("File copied successfully\n");`
	`    fclose(fp1);`
	`    fclose(fp2);`
	`}`
		[10]	**CO5**	**L3**

8	**Explain the different ways to detect END-OF-FILE with an example.** ● **explanation (8)** ● **program(2)** In Text File :	**[10]**	**CO5**	**L2**

•Special Character EOF denotes the end of File
•As soon as Character is read, End of the File can be detected.
•EOF is defined in stdio.h
•Equivalent value of EOF is -1

In Binary File :
• feof function is used to detect the end offile
• It can be used in textfile
• feof Returns TRUE if end of file isreached

Syntax :
int feof(FILE *fp);

<u>program</u>

```
#include <stdio.h>
int main () {
FILE *fp;
int c;

fp = fopen("file.txt","r");
if(fp == NULL)
{
perror("Error in opening file");
return(-1);
}
while(1)
{
c = fgetc(fp);
if( feof(fp) )
{
break ;
}
printf("%c", c);
}
fclose(fp);
return(0);
}
```