CSO-101 END SEMESTER ANSWER SHEET

NAME-PARTIK SINGH BUMRAH

BRANCH-COMPUTER SCIENCE AND ENGINEERING

ROLL NO-21075064

SECTION-BA1

EMAIL ID- partik.sbumrah.cse21@itbhu.ac.in

1.)
```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

struct student {
    char name[11];
    int id, fee, tuel;
};

int main()
{
    struct student a[50];
    for(int i=0; i<50; i++)
    {
        a[i].id = ((rand()) + 10000) % 1000000;
        a[i].fee = (rand() % 301);
        a[i].tuel = (rand() % 101);

        for(int j=0; j<10; j++){
            a[i].name[j] = (rand()%26) + 97;

        a[i].name[10] = '\0';
    }

    printf("\n Student Name    Student Id    Scholarship\n");
```

7 for (int i=0; i<50; i++);

{ int sel = 0;

if (a[i].jee >= 200 && a[i].jee <= 300)

{ if (a[i].tuel >= 90 && a[i].tuel <= 100)

sel = 100;

else if (a[i].tuel >= 80 && a[i].tuel < 90)

sel = 75;

else if (a[i].tuel >= 70 && a[i].tuel < 80).

sel = 50;

}

else if (a[i].jee >= 100 && a[i].jee <= 200)

{ if (a[i].tuel >= 90 && a[i].tuel <= 100)

sel = 75;

else if (a[i].tuel >= 80 && a[i].tuel < 90)

sel = 50;

else if (a[i].tuel >= 70 && a[i].tuel < 80)

sel = 25;

}

printf("|%-10s | %-10.5d | %-10d |\n"
, a[i].name, a[i].id.sal);

return 0;

}

7

2. Sol

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

int main () {
    File *fptr;
    char w1[50], w2[50];
    int n; int count = 0;
    printf("Enter the word to be replaced: ");
    scanf("%s", w1);
    printf("Enter the word that will replace: ");
    scanf("%s", w2);
    printf("Enter the number of times the word is replaced: ");
    scanf("%d", &n);

    int len = strlen(w1);
    char str[5000];
    fptr = fopen("File.txt", "r");

    int k = 0;
    char c = fgetc(fptr);
    while (c != EOF) {
        str[k] = c;
        ++k;
        c = fgetc(fptr);
    }

    fclose(fptr);
```

```
fptr = fopen("File.txt","w");
for(int i=0; i<k; i++)
{ if(n<0 || count<n){
    for(int j = 0; j<len; j++){
      if(str[i+j]==W1[j]){continue;}
      else{goto label;}
    }
fprintf(fptr,"%s",W2);
i+=len;
count++;}
label:
fputc(str[i],fptr);
}
if(count!=n && n>=0){printf("The word
if(count!=n && n>=0){
printf("The word is replaced %d time",count);
fclose(fptr);
return 0;
}
```

34

Storage classes provides us information about the location, visibility and life-time of a variable. The storage class decides the location of the program where the variables are recognised.

The different types of storage classes in C are :-

1. auto - auto is the default storage class of every variable declared inside a function or a block. These variables can only be accessed within the function / block in which they are declared. By default auto variables contains a garbage value. Unless they are initialised explicitly.

2. static - static variables also can be accessed within a function / block in which they are declared. static variables have an interesting property of retaining their only declared value. So they are initialised once and exist till the program terminates, and as no new value is allocated to it, hence no new memory is allocated as well. By default it contains a value zero until initialised.

3. extern. - extern storage class simply tells us that the variable is declared with a global scope ie the variable declared in a function can be accessed or modified in any other function as well. 'extern' keyword is used to declare a extern variable. The main purpose of extern variables is that they can be used at in any function in a program or

can be accessed between two different files which are part of a large program. By default these are assigned a value 0.

4) Register - register variable are same in terms of functionality and scope as auto variables, only difference is that these are tried to store in the register of the microprocessor if a free registration is available. This makes the use of a register variable much faster during runtime but if free registration is not available then these are also declared in the memory only. Also we can't access the address of these variables using pointers.

b) The difference between where the preprocessor searches for the file to be included in the code.

The two types is in the location

#include <filename>
#include "filename"

#include <filename>

It is generally used for pre-defined header files.

Eg:- #include <stdio.h>
#include <string.h>
#include <stdlib.h>

The preprocessor searches in an implementation dependent

manner normally in search directories pre-designated by the compiler, i.e. the compiler only search the locations where standard library headers are residing. The header files can be found at default locations like /usr/include or /usr/local/include.

#include <stdio.h>

#include "filename".

#include" " is used for header files that are written by a programmer himself. The preprocessor searches for the ~~already~~ file in the current directory ~~and also in the search directories pre defined by the compiler~~.

#include < >	#include " "
1.) The preprocessor searches in the search directories designated by the compiler.	1.) The preprocessor searches in the same directory as the file containing the directive.
2.) The header files can be found at default locations like /usr/include or /usr/local/include	2.) The header files can be found in - I defined folders.
3.) This method is normally used for standard library header files.	3.) This method is normally used for programmer-defined header files.

4) {
a) #include <stdio.h>
int main() {

char x;
printf("Enter the character to find its ASCII value.");
scanf("%c", &x);

int f = (int) x;

printf("The ASCII value of %c is %d", x, f);

b) The enumerated data is a user defined data and it gives, use the liberty to ~~~~ invent our own data type and define what values the variable of this data type can take. This data type helps in making the program listings more readable.
An enum can be defined as.
Eg:- ~~int~~
enum mar_status
{ single, married, divorced, widowed } → (1)
};
enum mar_status person1, person2; } → (2).

a) (1) declares the data type and specifies its possible values. These values are called 'enumerators'.
b) (2) declares the variables of this data type.

Now we can assign values to these enum variables.

→ person1 = married;
person2 = single;

Note that the values that weren't in the original declaration can't be assigned to the variables.

Thus an expression like

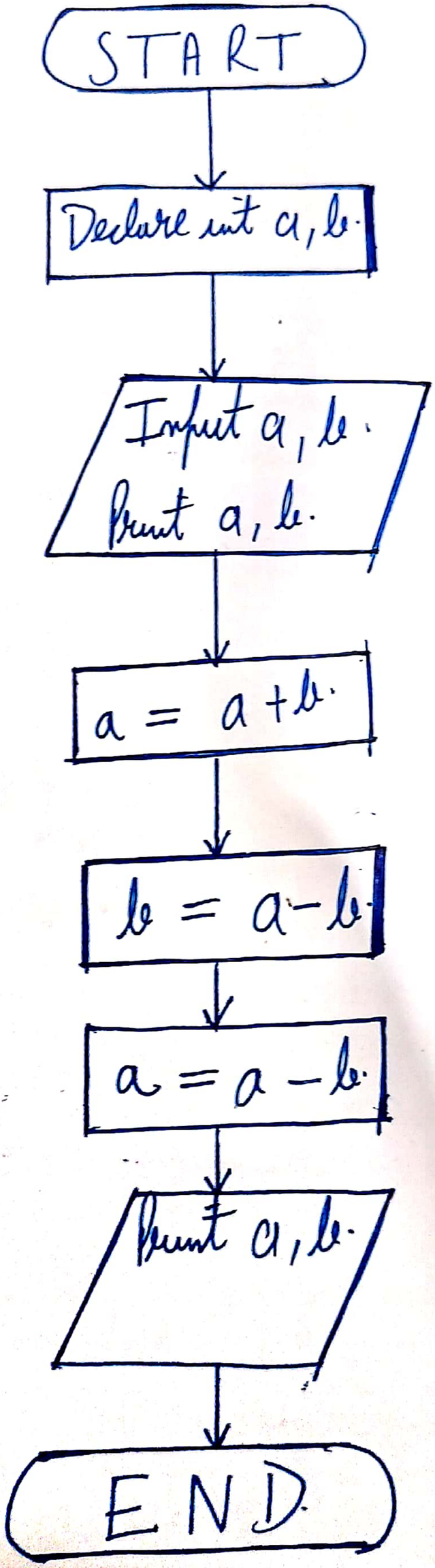
person3 = unknown;

will throw an error.

Also, the compiler internally treats the enumerators as integers. Each enumerator corresponds to an integral value starting from 0. So in the example, single is stored as 0, married is stored as 1, divorced as 2 and widowed as 3.

5.6 Algorithm.

Step 1: Start
Step 2: Declare int a, b.
Step 3: Input a, b.
Step 4: Print a, b.
Step 5: a = a + b
Step 6: b = a - b.
Step 7: a = a - b.
Step 8: Print a, b.
Step 9: End.



6 Nested switch case.

```
#include <stdio.h>
int main() {
int x=1, n=2;
switch (x) {
case 1:
    switch (n) {
    case 1:
        printf("1 1");
        break;
    case 2:
        printf("1 2");
        break;
    case 3:
        printf("1 3"); break;
    }
    break;
case 2:
    printf("2 2"); break;
}
return 0;
}
```

In the switch case statements the case labels are used to identify each case explicitedly. The case label ends with a colon and each is associated with a block of statements.

In case same case label occurs in one switch statement, it will throw an error. But it is not necessary for inner and outer case labels to be distinct in case of nested switch statements.

The inner case label works on the switched value of inner switch (x in case of the example) while the outer switched value works on the outer block itself so distinct values need not be used for inner and outer switch case.

Also in the example it is also shown that case label case 1: is used in both inner and outer switch statement but still the program ~~runs~~ was working fine.

7

7.4 Bitwise Operator	Logical Operator
1. It is the type of operators provided by the programming language to perform computation.	1. Logical operator is a type of operator provided by the programming language to perform logic based operations.
2. Bitwise operator work on bits and perform bit by bit operations.	2. Logical operators are used to make a decision based on multiple condition.
3. Bitwise operator operates on integers (on bits) and return a integer as a result.	3. Logical operators operate on booleans and also return booleans as a result.
4. Bitwise operators are &, |, ^, <<, 1, >>.	4. Logical operators are &&, ||, !.
5. Used in bit manipulation	5. Used in loops and condition statement.

Eg:-
```
#include <stdio.h>
int main()
{
    int a = 4, b = 5;
    printf("%d %d", a&b, a&&b);
    return 0;
}
```

Output → 1 4.

Explanation

Zero is considered as false and any other non zero integer as true. Thus in the first case (true && true) evaluates to true hence 1 is obtained as output. And the fuction and '&' works on bits and then returns the integer corresponding to the resultant bits.

8)

Given :-

```
main ()
{
float a = 1.3, b = 0.4;
printf ("%d", a % b);
}
```

This program will result in a compilation error.
as- ~~float~~

(%) → Modulo operator is an arithmetic operator which is used to obtain the remainder of an integer division. The modulo operator can't be applied to floating - point number i.e float or double ; if we will try to use modulo operator on floating point numbers it will result in a compilation error.

As both a and b are floating point i.e float variable, hence modulo operation on them will produce a compilation error!

a) Formatting characters or format specifiers are used during input and output operations in C. It is basically a way to tell the compiler what type of data we are taking input (using scanf()) and printing (using printf()). %d, %f, %c etc are some examples of format specifiers or formatting characters.
If we will use wrong formatting characters in the printf function then some unexpected result will occur.

b) If we use float for an integer.

If we will use the float format specifier ie "%f" for printing an integer in the printf() function then 0.000000 will be printed for any integer.

Eg:- #include <stdio.h>
int main(){

int x;
scanf("%d", &x);
printf("%f", x);
}

This program will first take an input x as an int variable and will print 0.000000 for any value of x.

2.7 If we use integer for a character

If we will use the int format specifier ie "%d" for printing a character then the ASCII value of the character will be printed. In C ASCII is a character encoding scheme which stores the characters as numerical value.

Eg:- #include <stdio.h>
int main ()
{
char x = 'a';
printf ("%d", x);
}

This program will print the ASCII value of the character stored in x ie 'a' which is equal to 97. So 97 will be the output of this program.

10.4.

```
#include<stdio.h>
#include<math.h>
int main()
{
    float a, b, c, disc, root1, root2, real, img;
    printf("Enter the values of a, b and c : ");
    scanf("%f %f %f", &a, &b, &c);
    disc = b*b - 4*a*c;
    if (disc > 0)
    {
        root1 = (-b + sqrt(disc)) / (2*a);
        root2 = (-b - sqrt(disc)) / (2*a);
        printf("root1 = %f and root2 = %f", root1, root2);
    }
    else if (disc == 0) {
        root1 = root2 = -b/(2*a);
        printf("root1 = root2 = %f", root1);
    }
    else {
```

```
        real = -b/(2*a);
        img = sqrt(-disc)/(2*a);
        printf("root1 = %.2f+%.2fi and root2 = %.2f-%.2fi",
                real, img, real, img);
    }
    return 0;
}
```

11.

```
#include "stdlib.h"
void update(int *pa, int *pb);
{
    int a;
    a = *pa;
    *pa = a + *pb;
    *pb = abs(2a - *pb);
}
int main()
{
    int a, b;
    int *pa = &a, *pb = &b;
    scanf("%d %d", &a, &b);
    update(pa, pb);
    printf("%d\n%d", a, abs(b));
}
```

11.7. #include "stdlib.h"
void update (int *pa, int *pb)

{
int a;

a = *pa;
*pa = a + *pb;
*pb = abs(a - *pb);

}

int main(){
int a, b;
int *pa = &a, *pb = &b;
scanf("%d %d", &a, &b);
update(pa, pb);
printf("%d\n%d", a, abs(b));

return 0;
}

The above program used the pass by reference method of passing values to a function.

In this program there is a function update which does not return anything and takes 2 pointer *pa and *pb as input.

We need to set the value of a as the sum of a+b and the value of b as absolute difference of a, b.

So:

In the function update we will first declare a int variable which will store the current value towards which for is pointing i.e. (i)

The we will assign the value towards which *fa is pointing to (a + *fb) where *fb represents the current value towards which fb is pointing.

ie *fa = a + *fb —— (1)

And they we will assign the value towards which fb is pointing to abs(2*a - *fa).

ie abs (2*a - *fa). —— (2)

(2) is equivalent to the absolute difference of the initial p value towards which fa and fb were pointing as.

a stores the initial value of *fa and now from (1) *fa = a + *fb.

So (2) will be equivalent to

*fb = a - *fb in terms of the initial values.

Note: In the main function we had ~~assigned~~ ~~int~~ declared 2 int variables a, b and we are taking input from the user for them. Then two pointers are initialized ~~to~~ which are *fa and fb

and *fa stores the address of a and fb stores the address of b..

Then the update function is called and fa and fb are passed as arguments and the above stated function is carried out and as the address of both a and b was passed hence the change of values will happen in values of the original variables ie a, b.

Hence ~~a will~~ assumed a value
a will be equal to a+b
and b will be assigned a value equal to abs(a-b).

Hence when the value are printed the output is the sum and absolute difference of the initial a and b.

12.

code:

```
#include <stdio.h>
int main() {
int a[4][2] = {{1,2},{3,4},{5,6},{7,8}};
int i, j;
for (i=0; i<4; i++) {
    for (j=0; j<2; j++) {
        printf("%d", *(*(a+i)+j));
    }
    printf("\n");
}
return 0;
}
```

Output

```
1 2
3 4
5 6
7 8
```

Explanation:

Here we are using two nested for loops to iterate through all the elements of the 2-D array. We know that for a 1-D array ~~Array~~ the name of the array is basically a pointer which points towards the first element of the array or stores the base address of the array. By using pointer addition with a number we can iterate through all the elements of the array i.e if x is a 1-D array. then x+i is the address of the element which is i th from the base element or is the i th index

so $(x+i) \equiv \&x[i]$

$*(x+i) = *(\&x[i])$

$*(x+i) = x[i]$ —— (1).

And each row of a 2-D array can be considered as a 1-D array., so a 2-D array can be considered as a collection of 1-D arrays that are placed one after another. In case of 2-D array expressions like a[0] or a[1] —— refers to the base address of the zeroth, first equivalent 1-D array.

So to ~~add~~ access an element that is the j th element from the base address of the i th 1-D array (equivalent) we can use

$a[i][j] = *(a[i]+j)$.

$= *(*(a+i)+j)$

This is what we used to print the elements of the

array in the above stated example.

13-4 The program will result into an error.

Explanation

In the code functions are declared inside another function. C does not support the Nesting of functions. C50() and end5en() are two functions defined in the main() function. Such nested function definition can not access local variables of the surrounding blocks, they can access only global variables of the containing module. This is done so that lookup of global variables does not have to go through directory. As in C, there are two nested scopes: local and global. Therefore nested functions have on a limited use. So approaching a nested function in C gives a compilation error.

14.

```
#include <stdio.h>
int board[1000][1000];
void f2(int r, int a, int n){
    if(r>=n){return;}
    else{board[r][a]=1;
    a= (a==0)?2:0;
    return f2(r+1,a,n);}
}
void f1(int n){
    if(n<=2){return;}
    else if(n>=3){f2(0,0,n);}
}
int main(){
    int n;
    printf("Enter the value of n:");
    scanf("%d",&n);
    f1(n);
```

```
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        printf("%d", board[i][j]);
    }
    printf("\n");
}
return 0;
}
```

15)

Code

```
#include <stdio.h>
void swap (int *a, int *b){
    int c = *a;
    *a = *b;
    *b = c;
}
int main (){
    int a, b;
    int *x = &a, *y = &b;
    scanf("%d %d", x, y);
    printf("%d %d", a, b);
    return 0;
}
```

Explanation :-

Here we have created two pointers x, y pointing to a and b respectively. On calling swap function we pass x and y as arguments. In other words

we are passing address of a and b to it, so the function can update the values ~~to the~~ of a and b using this address. Using simple arithmetic logic we have swapped the values of the two variables by storing the initial value of a in a temporary variable then updating the value of *a or a through x to *b or b and then assigning * b the temporary variable or the initial value of a.

16. The output of the program is jkl.

Output = jkl

In this program we have used a pointer array to store an array of strings. Each element of this array points towards a string.

ptr() is a function which have a double pointer and we have passed the address of the first string of *argv[].

So

p = "abc" t first string]

now t = (p + sizeof(int)) [-1]

and sizeof(int) = 4

so t = (address of "abc") + 4) [-1]

t = (address of "mno") [-1]

and this will be equivalent to

t = (address of "ojkl").

Hence the program print ojkl.

If f is printed it displays the address of "mno" string.

as f is a character double pointer.

17)

```
for (i = 2; i <= sqrt(n); i += 3)
    printf ("%d", i);
```

Sol n)

```
int i = 2;
while (i <= sqrt(n))
{   printf ("%d", i);
    i += 3;
}
```

18. Loop indices → 10 9 8

Number of iterations → 7

Initially i = 1, j = 1, k = 1 and count = 0.

so the compiler enters the while loop and for loop as both conditions come out to be true true — then inside the do while loop. k is increment. to k += i and i = 1 and the condition is k < 8 so with every iteration k will be incremented by i till k is not equal to 8 so the loop do while loop will be iterated 7 times hence count becomes equal to 7. and k = 8.

then as j is to get incremented by j += k so j becomes 1 + 8 = 9. and compiler gets out of the for loop. then i get increment by j = 9. so i becomes 10.
and then the compiler also exits the while loop.

19. Output of the program

int a[] = {22, 19, 17, 36, 12, 15, 28, 35, 66, 43};

elements = 10.

so, size of (a) = 10 * sizeof (int)

so. n = 10

the code inside if (a[i] > a[j]) is used to swap the values of the two terms.

let a[i] = a and a[j] = b.

then

a[i] = a[j] + a[i] (so a[i] = a+b).

a[j] = a[i] - a[j] (Now a[j] = (a+b-b) a[i] = a)

a[i] = a[i] - a[j] (a[i] = (a+b) - a = b) = a)

so finally a[i] = b and a[j] = a

Using both the nested loops if any element as i^th index is greater than the element with a j^th index will be swapped, hence eventually the array will be sorted, hence eventually the ~~will~~ the 1st element with the largest value will be at a[0] and smallest will be at the last position.

Hence the array will be reversely sorted so output will be -

66, 43 36 35 28 22 19 17 15 12.

20). The while loop will be executed 25 times

as a++ will increment the ASCII value by 1 so initially a was having ASCII value of 98 ie b and at the end it will have the value 123 after 'z'.

Hence loop will be iterated 25 times.

21. The code will produce a compilation error as the declaration of x, y is missing.

But if ~~it~~ we suppose that it was declared then the final value of x will be -50.

as (x > y) evalutes to false.

(y < 0) also evalutes to false.

hence else ~~condition~~ will be executed.

22. The program will produce an output of.

96
102

as y is declared as a static variable so during first func() call x will be incremented to 6 and y to 16. so x * y = 96 will be printed.

During the second call y will retain its initial value i.e 15. so it will be incremented to 16 and x will be incremented to 7 from 6 so 7 * 16 = 102 will be printed.

23. The output of the program will be.

```
* * *
* *
*
```

During the execution count = (3) for the first time so "*" will be printed 3 time during 1 row then new line will be added.

Then 2 & "*" will be printed and as row = 1 hence during last execution 1. "*" will be printed.