CMR
INSTITUTE OF
TECHNOLOGY

USN | | | | | | | | | | |

Internal Assessment Test 1

Sub:	DATA STRUCTURES AND APPLICATIONS				Code:	15DS33
Date:	7/09/2016	Duration: 90 mins	Max Marks: 50	Sem&Section: 3D	Branch:	CSE

Note: Answer any 5 questions.

Q1. a) Define Data Structure. Explain the Classification and Operations of Data structure.[1M+4M]
b) Differentiate between Structure and Union.[5M]

Q2.Write C Program to search key element in an array using BINARY SEARCH.(support with user Defined Function)[10M]

Q3. a)Write a C program to Create a structure called BANK with the fields Bank_name(string), type_of_bank(string), repo_rate(float),Reverse_repo_rate(float),interest_rate(float).Accept the data for each field and display with the suitable message.[5M]
b) Define pointer. How are pointers declared and initialized? Write a C program to Swap two numbers using pointers and functions.[5M]

Q4.Define Dynamic Memory Allocation. Explain the dynamic allocation functions with syntax and example.[10M]

Q5. Define STRING. Explain the following string built in functions along with syntax and example
a)strcmp b)strcpy c)strchr (1M+9M)

Q6.Define an ARRAY. Write a Menu driven C program to perform following operations on Array:
a)insert
b)delete
c)accept
d)display
Support each of the above operations with functions.

Q7.a) How is one dimensional-array allocated dynamically? Explain with an example. [5M]
b) Define Structure. How are Structure variables Declared and Initialized?[5M]

Q1. a) Define Data Structure. Explain the Classification and Operations of Data structure.[1M+4M]

Data Structure is way of Organizing data in a memory so that it can be used efficiently.

or

Data Structure is Specialized format for Organizing, Storing and retrieving the data.

Types of Data Structure

* Data Structure can be classified as follows
  1) Primitive Data Structure
  2) Non-Primitive Data Structure
  3) Linear Data Structure
  4) Non-Linear Data Structure

(a) Primitive Data-type or Data Structure

* Basic data types that are available in most of the Programming Language. Data structures that are directly operated upon by machine-level instructions are known as primitive data types.

* For example
  1) Integer (int): used to represent a number without decimal point Eg: 12, 90
  2) Float or double (float): used to represent a number with decimal point Eg: 12.50
  3) Character (char): used to represent single character Eg: 'c', 'a'
  4) Boolean: used to represent logical values i.e true or false

Non-Primitive Data Structure

* The data structures are derived from the primitive data structures. They stress on formation of sets of homogeneous and heterogeneous data elements.

Example: Arrays, Stack, Queue, Linked List, Tree, Graph

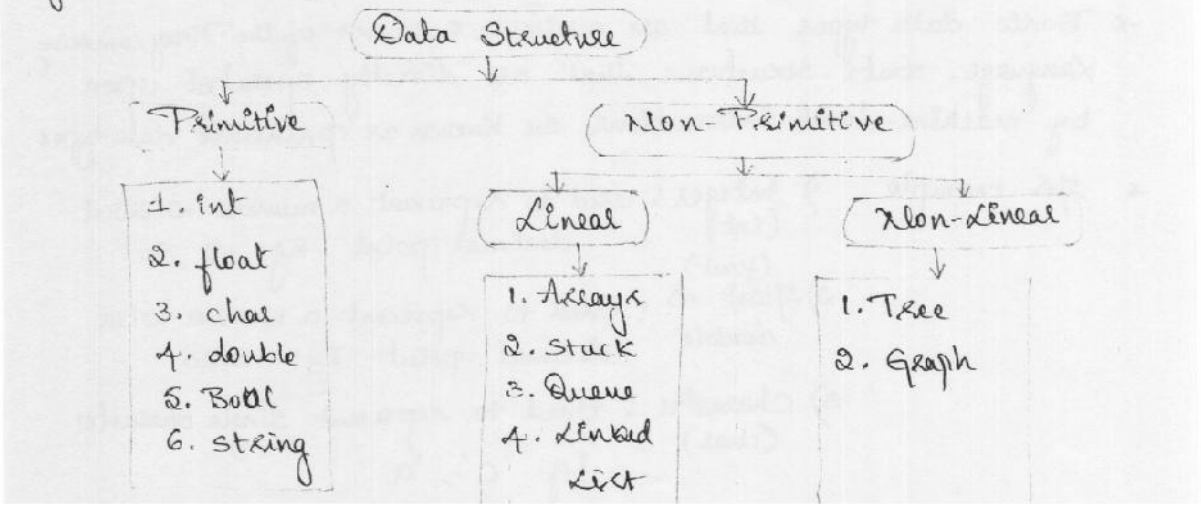
Linear - Data Structure

* In Linear Data Structure, data is arranged in linear fashion. Eg: Arrays, Stack Queue, Linked List

Non - Linear Data Structure

* In Non Linear Data Structure, data is not arranged in order or linear fashion

Eg: Trees, Graph, table etc.



b) Differentiate between Structure and Union.[5M]

Difference Between Structure and Union

STRUCTURE	UNION
1) Keyword struct is used to define structure	1) Keyword Union is used to define a Union
2) When a variable is associated with a structure, the compiler allocates memory to each of the variable.	2) When a variable associated with a union the compiler allocates memory by considering the size of the largest memory.
3) The size of structure will be greater than or equal to the sum of sizes of its members	3) The size of union will be equal to size of largest member.
4) Each member with in a structure is assigned unique storage area of locations	4) Memory allocated is shared by individual members.
5) The address of each member will be in ascending order. This indicates that memory for each member will start at different offset values.	6) The address is same for all the members of a union. every member begins at different offset values.
6) Individual member can be accessed at a time	6) Only one member can be accessed

2).Write C Program to search key element in an array using BINARY SEARCH.(support with user Defined Function)[10M]

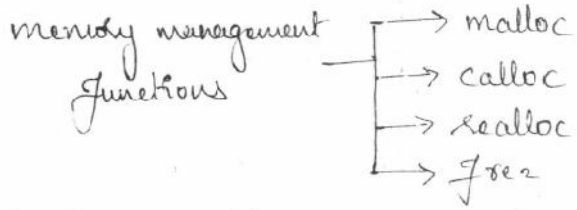
```c
#include <stdio.h>
void b_search(int a[20],int n,int search);


int main()
{
   int c, n, search, array[100];

   printf("Enter number of elements\n");
   scanf("%d",&n);

   printf("Enter %d integers\n", n);

   for (c = 0; c < n; c++)
      scanf("%d",&array[c]);

   printf("Enter value to find\n");
   scanf("%d", &search);

           b_search(a,n,search);
}
void b_search(int a[20],int n,int search)
{

 int first, last, middle;
   first = 0;
   last = n - 1;
   middle = (first+last)/2;

   while (first <= last) {
      if (array[middle] < search)
         first = middle + 1;
      else if (array[middle] == search) {
         printf("%d found at location %d.\n", search, middle+1);
         break;
      }
      else
         last = middle - 1;

      middle = (first + last)/2;
   }
   if (first > last)
      printf("Not found! %d is not present in the list.\n", search);


}
```

Q4. Define Dynamic Memory Allocation. Explain the dynamic allocation functions with syntax and example.[10M]

Allocation of the memory during runtime or execution time is known as dynamic memory allocation. Here memory which is allocated is not fixed.

Dynamic memory management functions

* The memory management functions that are used to allocate & dealloc. memory are shown below



* Using the functions malloc, calloc and realloc memory space can be allocated where as using the function free() space can be deleted.

① malloc()

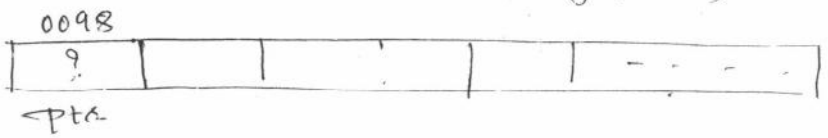
* Function allows the program to allocate memory explicitly as & when required and the exact amount needed during execution.
* This function allocates a block of memory. The size of the block is number of bytes specified in the parameter.
* It is defined in stdlib.h header file. Syntax is

```
#include<stdlib.h>
        :
ptr = (data type *) malloc(size);
```

where ptr is a pointer variable of type datatype
datatype can be any of the basic data type
Size is the number of bytes required.

For example: `int *ptr;`

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

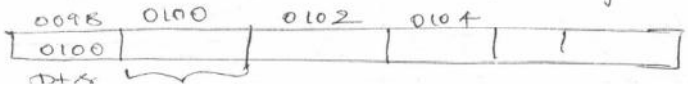


execute the statement

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

Since 2 bytes of free memory space is available, the function malloc allocates a block of 2 bytes of memory and returns the address of the first byte.

The returned address is stored in the pointer



- Calloc

* Calloc stands for contiguous allocation of multiple blocks and it mainly used to allocate memory for arrays.
* The number of blocks determined by n. The size of each block is equal to the number of bytes specified in the parameter i.e size.
* The total number of bytes allocated is n * size and all bytes will be initialized to 0.
* The syntax:

```
#include<stdlib.h>
.....
ptr = (data type *) calloc(n, size);
```

where

* ptr is pointer variable of type data-type
* data-type can be any of the basic data type or user defined data type.
* n is the number of blocks to be allocated
* size is the number of bytes in each block

* On successful allocation, the function return the address of first byte of allocated memory. Since the address is returned, the return type is a void pointer, By type casting appropriately we can use it to store integer float etc.
* If specified size of memory is not available, the condition is called overflow of memory. In such case, the function returns NULL.

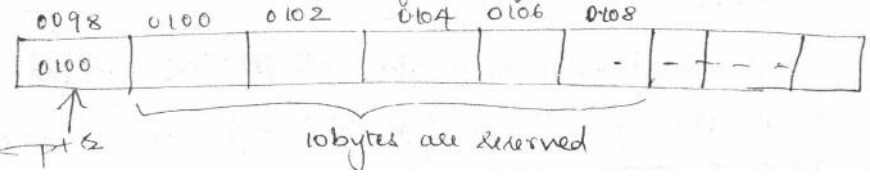
For example: `int *ptr;`

```
ptr = (int *) calloc( 5, sizeof(int));
```

execute the statement

```
ptr=(int *) calloc(5, sizeof(int));
```

Since 5*2=10 bytes of free memory space is available, the function calloc(), allocates 5 blocks of 2 bytes each, initialize each byte to 0 and returns the address of the first byte.



realloc

* Sometimes, the allocated memory may not be sufficient and we may require additional memory space. In another situation, where allocated memory may be much larger and we want to reduce.
* In both the situations, the size of allocated memory can be changed using realloc() and process is called reallocation of memory.
* realloc() changes the size of the block by extending or deleting the memory at the end of the block.
* If the existing memory can be extended, ptr value will not be changed.
* If the memory can not be ~~changed~~ extended, then function allocates completely new block.

Syntax: `#include<stdlib.h>`

`ptr = (datatype *) realloc(ptr, size);`

where

ptr is a pointer to block of memory which is allocated previously

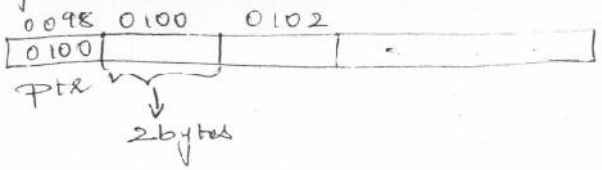
size is new size of block

* On successful allocation, the function returns the address of first byte of allocated memory.
* If memory can not be allocated, function returns NULL.

For example:

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

Here ptr is allocated with 2 bytes of memory.



* now if we want to extend memory, use realloc function

`ptr = (int *) realloc(ptr, 10);`

```
/* Program to show usage of realloc() function */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
Void main()
{
    char *str;
    Str = (char *) malloc(10);
    Strcpy(Str, "computer");
    Str = (char *) realloc (Str, 40);
    Strcpy (Str, "computer Science & Engineering");
}
```

free

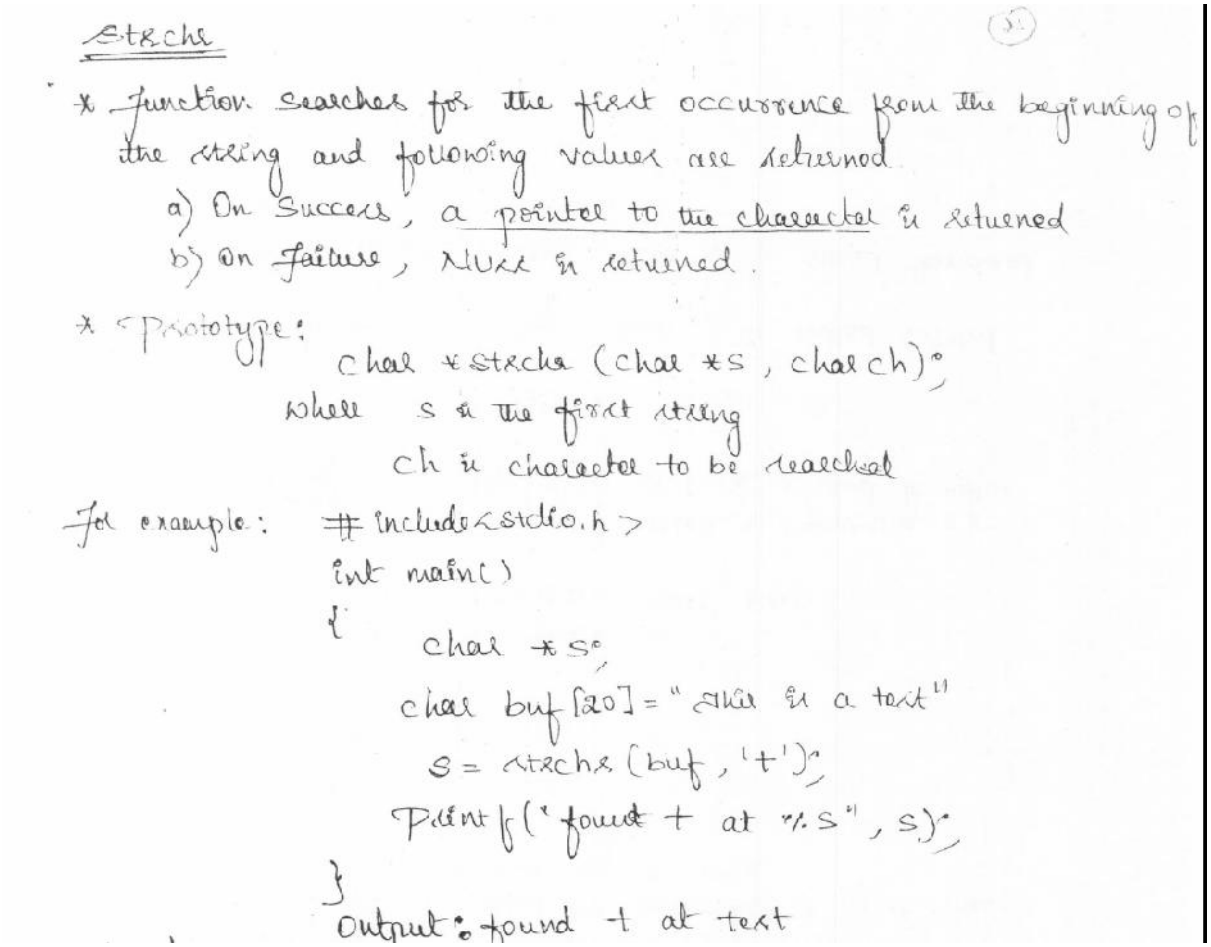
* This function is used to de-allocate the allocated blocks of memory which is allocated by using the functions calloc(), malloc() & realloc().
* It is responsibility of a programmer to de-allocate memory whenever it is not required by the program and initialize Ptr to NULL

Q5. Define STRING. Explain the following string built in functions along with syntax and example

a)strcmp b)strcpy c)strchr (1M+9M)

⇒ STRING

* An array (sequence) of characters is called String. A string is most conveniently represented using a sequence of storage locations in memory.



strcpy()

The C library function **char *strcpy(char *dest, const char *src)** copies the string pointed to, by **src** to **dest**.

Declaration

Following is the declaration for strcpy() function.

```
char *strcpy(char *dest, const char *src)
```

Parameters

- **dest** -- This is the pointer to the destination array where the content is to be copied.
- **src** -- This is the string to be copied.

Return Value

This returns a pointer to the destination string dest.

Example

The following example shows the usage of strcpy() function.

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

int main()
```

```
{
   char src[40];
   char dest[100];

   memset(dest, '\0', sizeof(dest));
   strcpy(src, "This is tutorialspoint.com");
   strcpy(dest, src);

   printf("Final copied string : %s\n", dest);

   return(0);
}
```

strcmp()

The C library function **int strcmp(const char *str1, const char *str2)**compares the string pointed to, by **str1** to the string pointed to by **str2**.

Declaration

Following is the declaration for strcmp() function.

```
int strcmp(const char *str1, const char *str2)
```

Parameters

- **str1** -- This is the first string to be compared.
- **str2** -- This is the second string to be compared.

Return Value

This function return values that are as follows:

- if Return value < 0 then it indicates str1 is less than str2.
- if Return value > 0 then it indicates str2 is less than str1.
- if Return value = 0 then it indicates str1 is equal to str2.

Example

The following example shows the usage of strncmp() function.

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

int main ()
```

```
{
  char str1[15];
  char str2[15];
  int ret;


  strcpy(str1, "abcdef");
  strcpy(str2, "ABCDEF");

  ret = strcmp(str1, str2);

  if(ret < 0)
  {
    printf("str1 is less than str2");
  }
  else if(ret > 0)
  {
    printf("str2 is less than str1");
  }
  else
  {
    printf("str1 is equal to str2");
  }

  return(0);
}
```

Q6.Define an ARRAY. Write a Menu driven C program to perform following operations on Array:

a)insert
b)delete
c)accept
d)display

Support each of the above operations with functions.

```
#include <stdio.h>
int n;
void display( int a[20])
{   int i;
        printf("the array elements are\n");
        for(i=0;i<n;i++)
                printf("%d\t",a[i]);
```

```
 }
void create(int a[20])
{   int i;
        printf("enter the array elements\n");
        for(i=0;i<n;i++)
                scanf("%d",&a[i]);
}
void insert(int a[20])
{
        int pos,i,val;
        printf("enter the position to be inserted\n");
        scanf("%d",&pos);
        printf("enter the value to be inserted");
        scanf("%d",&val);
        for(i=n-1;i>=pos-1;i--)
                a[i+1]=a[i];
        a[pos-1]=val;
        n++;
}
void delete(int a[20])
{   int pos,i;
        printf("enter the position to be deleted\n");
        scanf("%d",&pos);
        for(i=pos-1;i<n-1;i++)
                a[i]=a[i+1];
        n--;
  }

int main(void)
{
        int ch,a[20];
        for(;;)
        {
                printf("1:create \n 2:display \n 3:insert \n delete");
                printf("enter your choice\n");
                scanf("%d",&ch);
                switch(ch)
                {
                        case 1:printf("enter the number of elements");
                                scanf("%d",&n);
                                create(a);
                                 break;
                        case 2:display(a);
                                break;
                        case 3:insert(a);
                                break;
                        case 4:delete(a);
                                break;
                        default:printf("invalid choice ");
                                  return 0;
                }
        }
}
```

Q7.a) How is one dimensional-array allocated dynamically? Explain with an example. [5M]

b) Define Structure. How are Structure variables Declared and Initialized?[5M]

a)One-Dimensional Arrays

In C, arrays must have their extents defined at compile-time. There's no way to postpone the definition of the size of an array until runtime. Luckily, with pointers and `malloc`, we can work around this limitation.

To allocate a one-dimensional array of length *N* of some particular type, simply use `malloc` to allocate enough memory to hold *N* elements of the particular type, and then use the resulting pointer as if it were an array. For example, the following code snippet allocates a block of *N* `int`s, and then, using array notation, fills it with the values 0 through *N*-1:

```
int *A = malloc (sizeof (int) * N);
```

```
/*program to accept n number of elements using dynamic arr
#include<stdio.h>
#include <stdlib.h>
int main()
{
    int *a, i, j, n;
printf("enter the number of elements\n");
scanf("%d", &n);
    a = (int *) malloc (n, sizeof(int));
if(a==NULL)
{
printf("insufficient memory\n");
return;
}
printf("enter the array elements\n");
for(i=0;i<n;i++)
        scanf("%d", &a[i]);
printf("array elements are\n");
for(i=0;i<n;i++)
        printf("%d", a[i]);
}
```

b)

⇒ STRUCTURES

* Structure is a collection of one or more ~~data~~ one or more variables of same or different data types grouped together under the single name.

* Structure is a user defined data type that can store related information about an entity.

* Syntax: Struct Structure_name
{
data type var_name 1;
:
data type var_name 2;
};

* Initialization of structure is done after declaration

Struct Structure_name structure_variable_name;

for example: Struct person ⟵ name of the structure
{
char name[10]; ⎫
int age; ⎬ members of the structure
float salary; ⎭
};

Struct person P;
⟶ variable of structure

* Accessing the members of structure

* Inorder to access the members of the structure we require structure variable name and . operator

Syntax:

structure_variable . member
name name

* for example: inorder to store name of person

```
Strcpy(p.name, "xyz");
p.age = 10;
p.salary = 35000;
```

* The above can also be read dynamically using scanf

```
scanf("%d", &emp1.employee);
```

```
scanf("%s", p.name);
scanf("%d", p.age);
scanf("%f", p.salary);
```

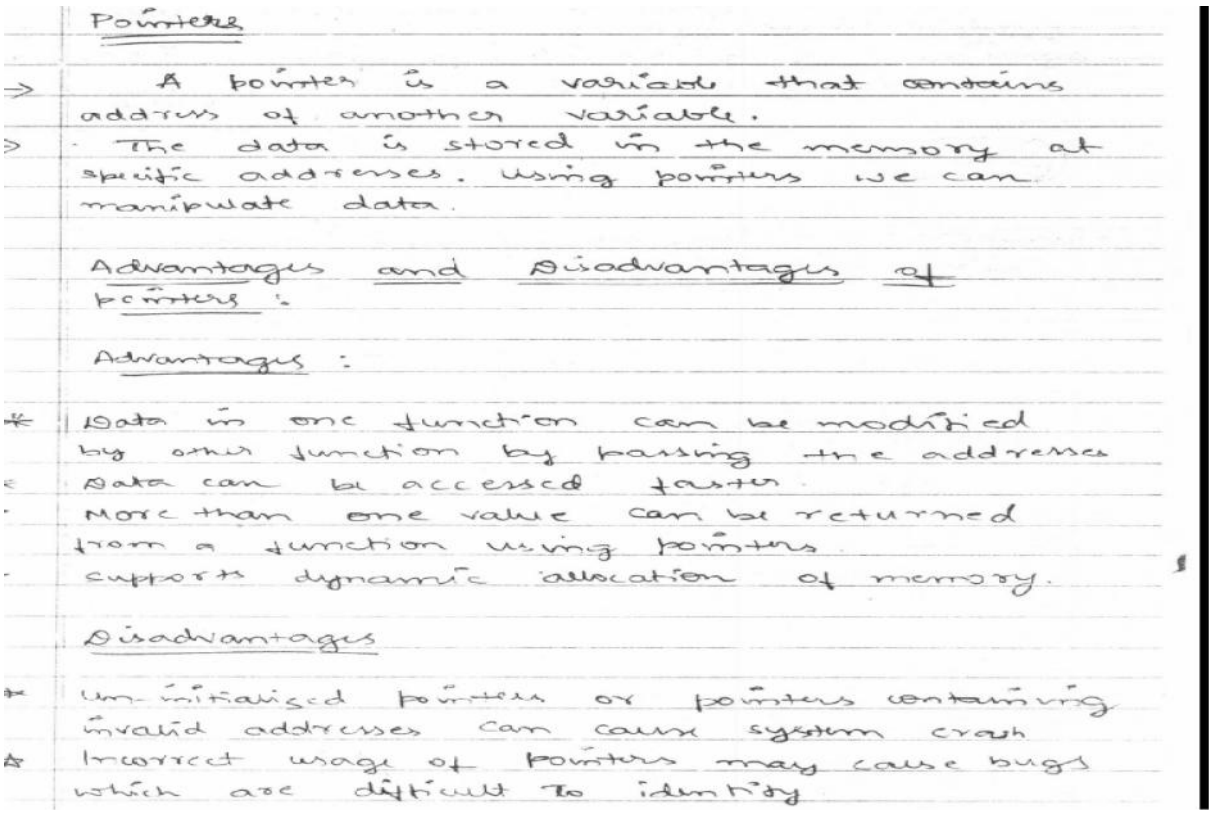
* C program to read an employee number, name, age salary and print details using structure */

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

struct employee
{
    int empnumber;
    char empname[20];
    int age;
    float salary;
};

void main()
{
    struct employee emp1;

    printf("enter employee number, name, age, salary\n");
    scanf("%d %s %d %f", &emp1.empnumber,
          emp1.name, &emp1.age, &emp1.salary);

    printf("employee number is %d\n", emp1.empnumber);
    printf("employee name is %s\n", emp1.empname);
    printf("employee name is %d\n", emp1.age);
    printf("employee name is %f\n", emp1.salary);

}
```

Q3. a)Write a C program to Create a structure called BANK with the fields Bank_name(string), type_of_bank(string), repo_rate(float),Reverse_repo_rate(float),interest_rate(float).Accept the data for each field and display with the suitable message.[5M]
b) Define pointer. How are pointers declared and initialized? Write a C program to Swap two numbers using pointers and functions.[5M]

```
#include<stdio.h>

struct BANK
{
        Char B_name[20],t_bank[20];
        Float repo,rev_repo,intr_rate;
}

struct BANK b;


main()
{
        printf("enter the details of the bank\n");
        scanf("%s%s%f%f%f",B_name,t_bank,&repo,&rev_repo,&intr_rate);
        printf("name of the bank is %s",B_name);
        printf("type of the bank is %s",t_bank);
        printf("repo rate of the bank is %f",repo);
        printf("reverse repo rate of the bank is %f",rev_repo);
}
```

b)



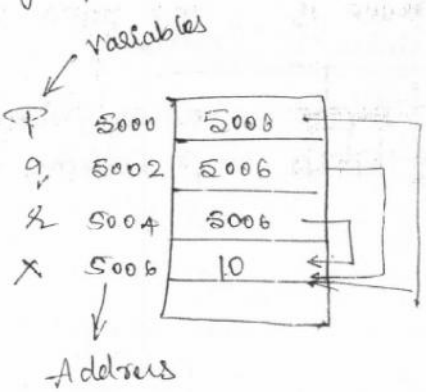
* Here the variable x is declared as integer data variable. Since p is a pointer variable of type integer, it should contain address of integer variable.
* Using the statement p = &x the valid address is stored in the pointer variable and hence the pointer variable p is initialized.

⇒ Accessing Variables through Pointers

* The value of a variable can be accessed using pointer variable using unary operator *. This operator is called indirection operator & dereferencing operator.

Consider the following declaration

```
x = 10;
r = &x;
q = &x;
p = &x;
```




```
printf("&p = %d, p = %d, *p = %d\n", &p, p, *p);
```

output :- &p = 5000, p = 5006, *p = 10

```
printf("&q = %d, q = %d, *q = %d", &q, q, *q);
```

output :- &q = 5002, q = 5006, *q = 10

```
/* Program to add two numbers using Pointers */
#include <stdio.h>
void main()
{
    int a, b, sum;

    int *pa, *pb
    pa = &a;
    pb = &b;

    printf("enter the value of a &b");
    scanf("%d %d", &a, &b); /* scanf("%d %d", pa, pb);
    sum = *pa + *pb;

    printf("sum = %d\n", sum);
}
```