Toic: pointer, structure and link list.

Pointer:

Question 1: Complete the function void update(int *a,int *b). It receives two integer pointers, int* a and int* b. Set the value of a to their sum, and b to their absolute difference. There is no return value, and no return statement is needed. [Add any libraries you want]

a'=a+b
b'=|a-b|

```
#include <stdio.h>
void update(int *a,int *b) {
    // Complete this function
}
int main() {
    int a, b;
    int *pa = &a, *pb = &b;

    scanf("%d %d", &a, &b);
    update(pa, pb);
    printf("%d\n%d", a, b);

    return 0;
}
```

Answer 1:

```
#include <stdio.h>
#include <stdlib.h>
void update(int *a,int *b) {
    // Complete this function
    int t1=*a+(*b);
    int t2=abs(*a-(*b)) ;
    *a=t1;
    *b=t2;
}
int main() {
    int a, b;
    int *pa = &a, *pb = &b;

    scanf("%d %d", &a, &b);
    update(pa, pb);
```

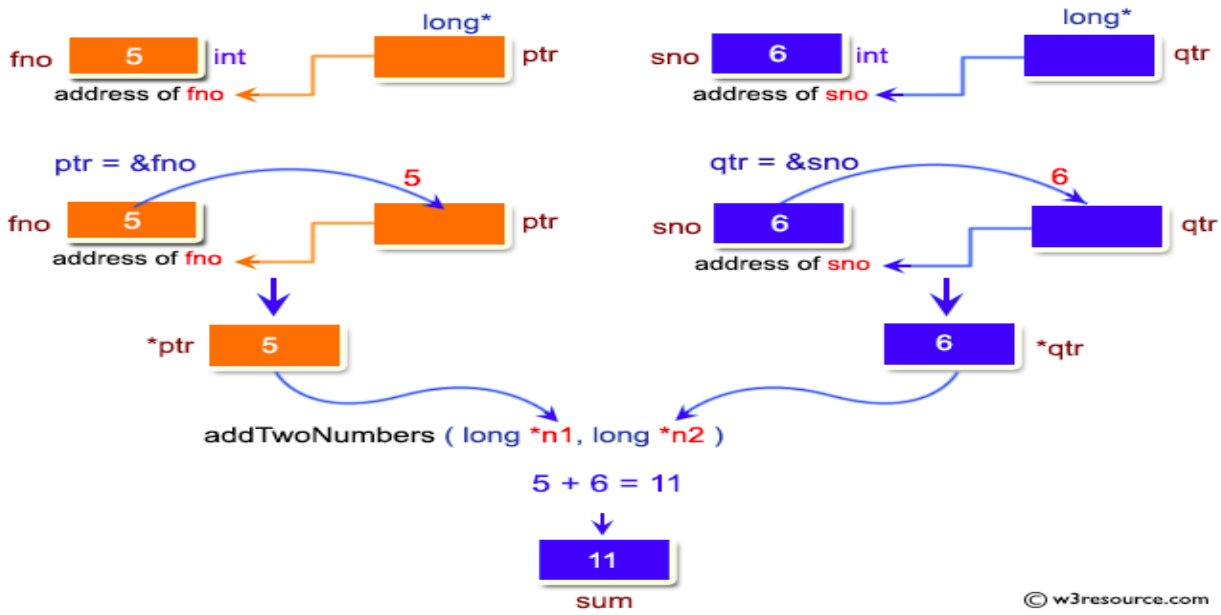
```
    printf("%d\n%d", a, b);
    return 0;
}
```

Question 2: Implement Call by reference using pointers to add two numbers.

Answer 2:

```
#include <stdio.h>
long addTwoNumbers(long *n1, long *n1){
   long sum;
   sum = *n1 + *n2;
   return sum;
}
int main()
{
   long fno, sno, sum;
   printf("\n\n Pointer : Add two numbers using call by reference:\n");
   printf(" Input the first number : ");
   scanf("%ld", &fno);
   printf(" Input the second  number : ");
   scanf("%ld", &sno);
   sum = addTwoNumbers(&fno, &sno);
   printf(" The sum of %ld and %ld  is %ld\n\n", fno, sno, sum);
   return 0;
}
```

Explanation:



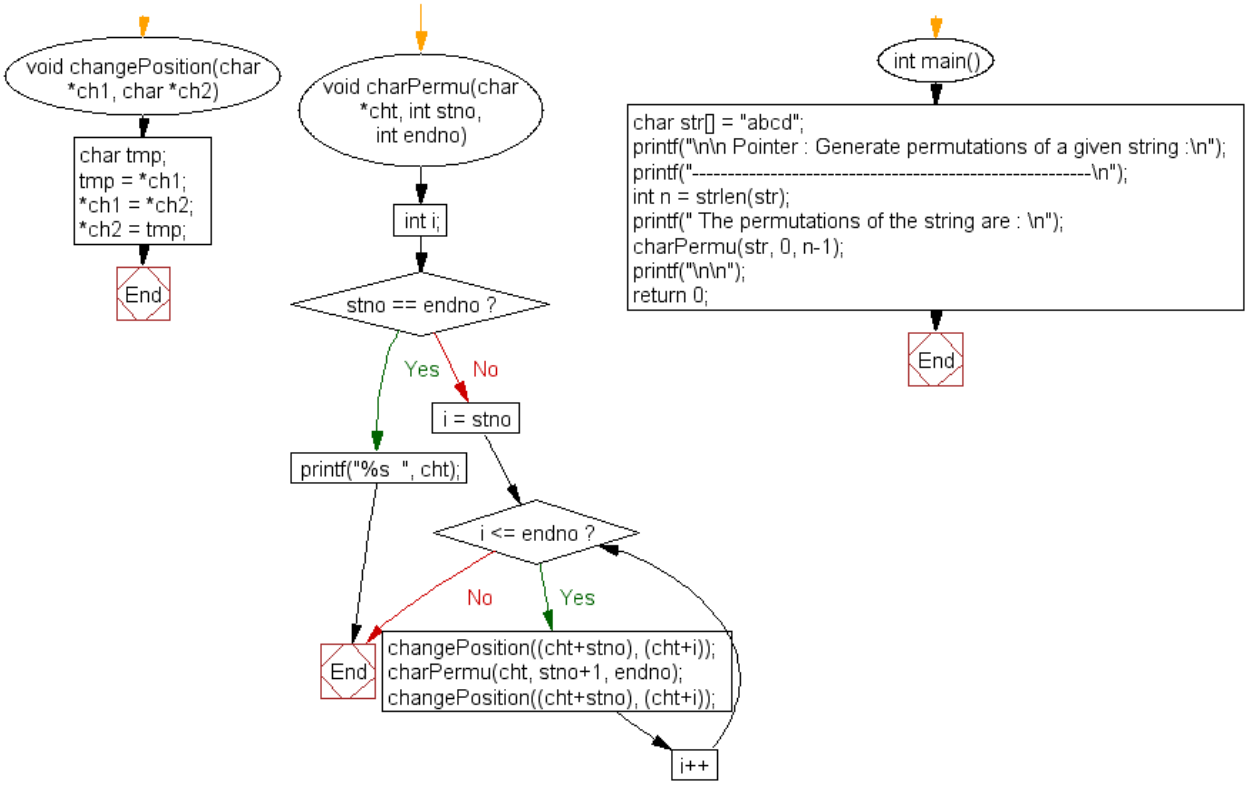
Question 3: Write a program in C to print all permutations of a given string using pointers.

Answer 3:

```
#include <stdio.h>
#include <string.h>
void changePosition(char *ch1, char *ch2)
{
    char tmp;
    tmp = *ch1;
    *ch1 = *ch2;
    *ch2 = tmp;
}
void charPermu(char *cht, int stno, int endno)
{
   int i;
   if (stno == endno)
     printf("%s  ", cht);
   else
   {
      for (i = stno; i <= endno; i++)
      {
         changePosition((cht+stno), (cht+i));
         charPermu(cht, stno+1, endno);
         changePosition((cht+stno), (cht+i));
      }
   }
}

int main()
{
    char str[] = "abcd";
   printf("\n\n Pointer : Generate permutations of a given string :\n");
   int n = strlen(str);
   printf(" The permutations of the string are : \n");
   charPermu(str, 0, n-1);
  return 0;
}
```

Explanation:



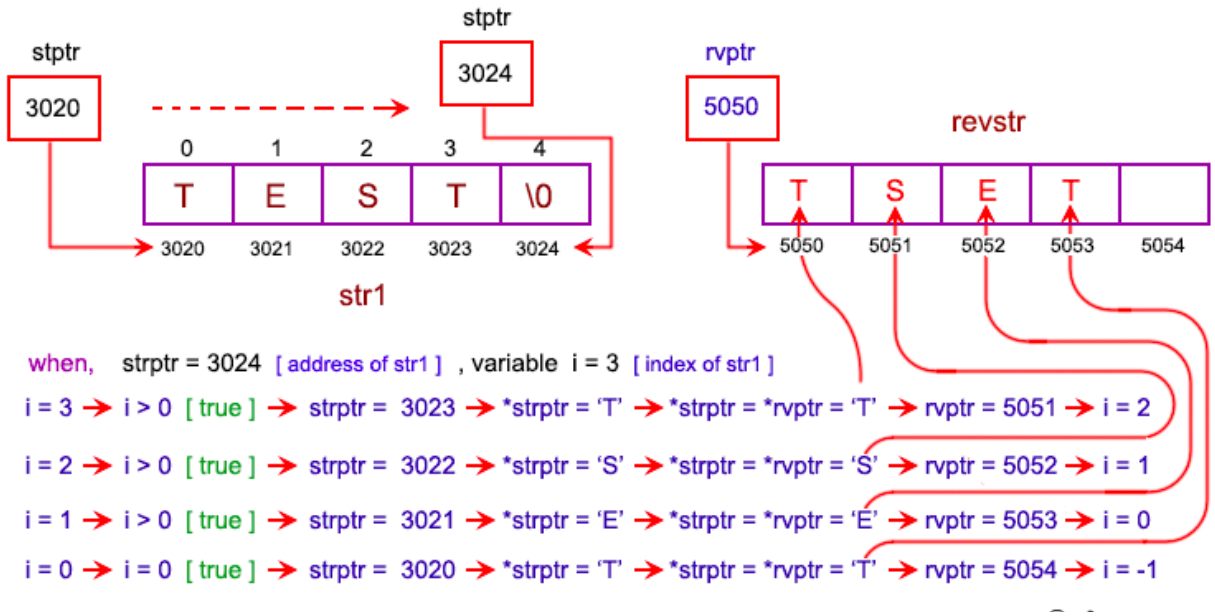
Question 4: Write a program in C to print all the alphabets using a pointer.

Answer 4:

```
#include <stdio.h>
int main()
{
   char alph[27];
   int x;
   char *ptr;
   printf("\n\n Pointer : Print all the alphabets:\n");
   ptr = alph;
   for(x=0;x<26;x++)
   {
      *ptr=x+'A';
      ptr++;
   }
   ptr = alph;
   printf(" The Alphabets are : \n");
   for(x=0;x<26;x++)
   {
     printf(" %c ", *ptr);
```

```
        ptr++;
    }
    return(0);
}
```

Explanation:



Question 5: Write a C program to add two matrices using pointers. C program to input two matrices from the user and find the sum of both matrices using pointers.

Answer 5:

```
#include <stdio.h>
#define ROWS 3
#define COLS 3
void matrixInput(int mat[][COLS]){
    int i, j;
    for (i = 0; i < ROWS; i++)
    {
        for (j = 0; j < COLS; j++)
        {
            scanf("%d", (*(mat + i) + j));
        }
    }
}
void matrixPrint(int mat[][COLS]){
    int i, j;
    for (i = 0; i < ROWS; i++)
    {
        for (j = 0; j < COLS; j++)
```

```
        {
            printf("%d ", *(*(mat + i) + j));
        }
        printf("\n");
    }
}
void matrixAdd(int mat1[][COLS], int mat2[][COLS], int res[][COLS]){
    int i, j;
    for (i = 0; i < ROWS; i++)
    {
        for (j = 0; j < COLS; j++)
        {
            // res[i][j] = mat1[i][j] + mat2[i][j]
            *(*(res + i) + j) = *(*(mat1 + i) + j) + *(*(mat2 + i) + j);
        }
    }
}
int main()
{
    int mat1[ROWS][COLS], mat2[ROWS][COLS], res[ROWS][COLS];
    printf("Enter elements in first matrix of size %dx%d: \n", ROWS, COLS);
    matrixInput(mat1);
    printf("\nEnter elements in second matrix of size %dx%d: \n", ROWS, COLS);
    matrixInput(mat2);
    matrixAdd(mat1, mat2, res);
    printf("\nSum of first and second matrix: \n");
    matrixPrint(res);
    return 0;
}
```

Structures and Unions

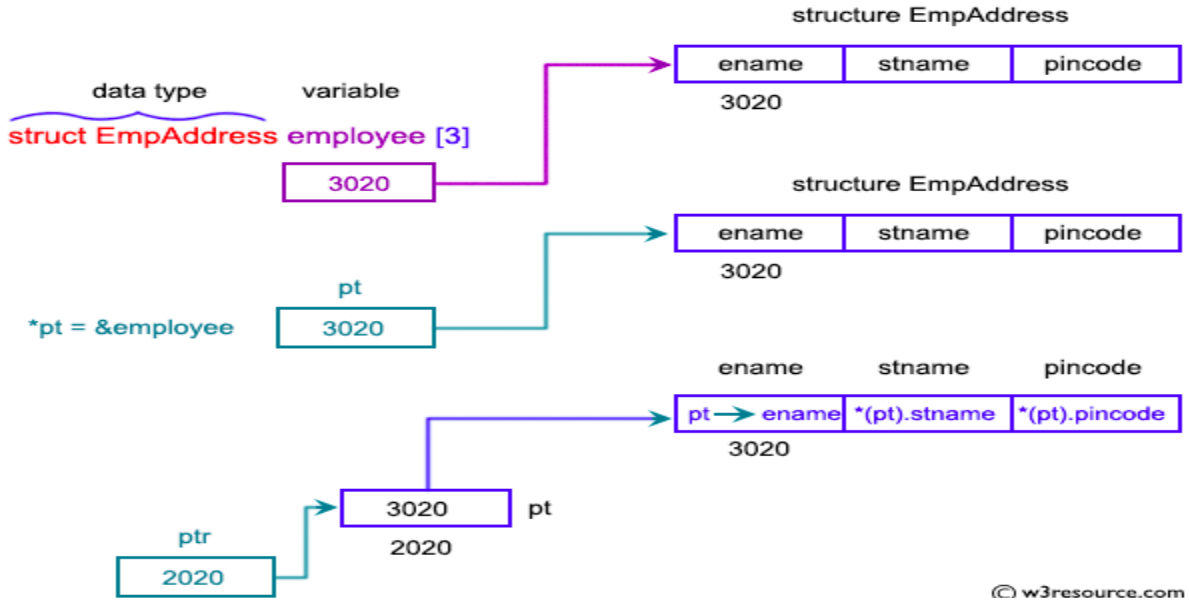
Question 1: Write a program in C to show the usage of pointer to structure.

Answer 1:

```
#include <stdio.h>
struct EmpAddress
{
char *ename;
char stname[20];
int pincode;
}
employee={"John Alter","Court Street \n",654134},*pt=&employee;
int main()
```

```
{
        printf("\n\n Pointer : Show the usage of pointer to structure :\n");
        printf(" %s from %s \n\n",pt->ename,(*pt).stname);
        return 0;
}
```

Explanation:

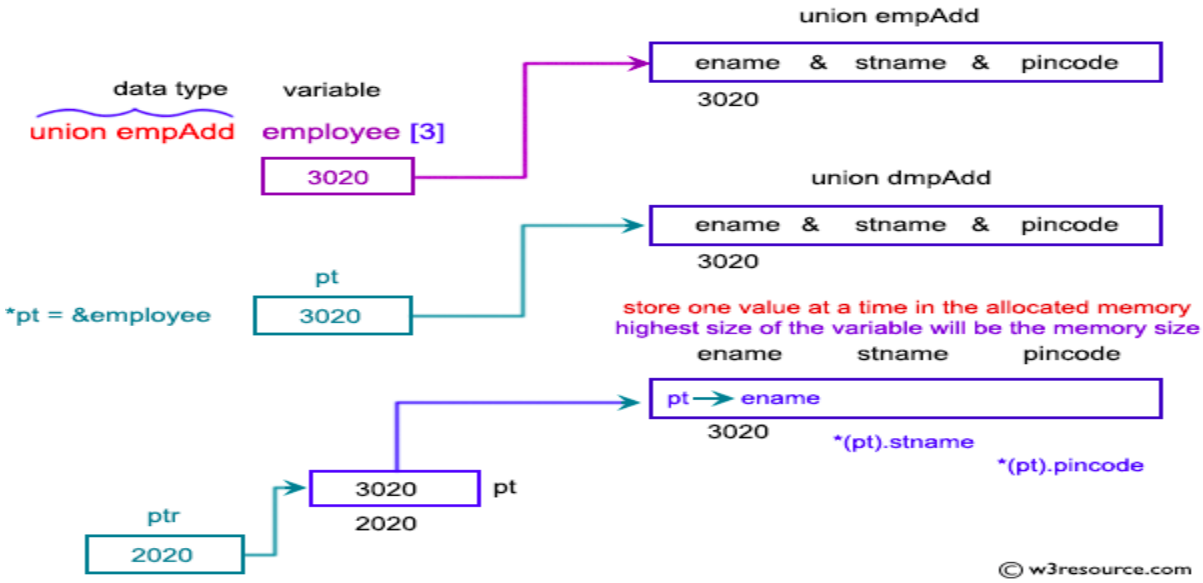


Question 2: Write a program in C to show a pointer to union.

Answer 2:

```
#include <stdio.h>
union empAdd
{
  char *ename;
  char stname[20];
  int pincode;
};
int main()
{
    printf("\n\n Pointer : Show a pointer to union :\n");
    union empAdd employee,*pt;
    employee.ename="Jhon Mc\0Donald";
    pt=&employee;
    printf(" %s %s\n\n",pt->ename,(*pt).ename);
    return 0;
}
```

Explanation:



Q3: What is output of the following C code:

```
#include<stdio.h>
struct st
{
    int x;
    struct st next;
};

int main()
{
    struct st temp;
    temp.x = 10;
    temp.next = temp;
    printf("%d", temp.next.x);
    return 0;
}
```

(A) Compiler Error
(B) 10
(C) Runtime Error
(D) Garbage Value

Answer: (A)

Explanation: A structure cannot contain a member of its own type because if this is allowed then it becomes impossible for the compiler to know the size of such a struct. Although a pointer of same type can be a member because pointers of all types are of same size and compiler can calculate size of struct

Linked List:

Q1:What is the output of the following function for starting pointing to the first node of the following linked list? **1->2->3->4->5->6**

```
void fun(struct node* start) {
        if(start == NULL)
                return;

        printf("%d ", start->data);

        if(start->next != NULL )
                fun(start->next->next);

        printf("%d ", start->data);
}
```

A. 1 4 6 6 4 1
B. 1 3 5 1 3 5
C. 1 2 3 5
D. 1 3 5 5 3 1

Ans: D

Explanation: fun() prints alternate nodes of the given Linked List, first from head to end, and then from end to head. If Linked List has an even number of nodes, then skips the last node

Q2. What does the following function do for a given Linked List with the

first node as head?

```
void fun1(struct node* head) {
        if(head == NULL)
                return;
```

```
    fun1(head->next);
    printf("%d ", head->data);
}
```

A. Prints all nodes of linked lists

B. Prints all nodes of linked list in reverse order

C. Prints alternate nodes of Linked List

D. Prints alternate nodes in reverse order

Ans 1 : B
Explanation: fun1() prints the given Linked List in reverse manner. For Linked List 1->2->3->4->5, fun1() prints 5->4->3->2->1.

Q3: Consider an implementation of an unsorted singly linked list.

Suppose it has its representation with a head pointer only.

Given the representation, which of the following operations can be

implemented in O(1) time?

i) Insertion at the front of the linked list

ii) Insertion at the end of the linked list

iii) Deletion of the front node of the linked list

iv) Deletion of the last node of the linked list

A. I and II

B. I and III

C. I, II and III

D. I, II and IV

Ans: B

Explanation:

For i, we are head and just need to update the next pointer of the head node which takes O(1) time.

For ii, we first need to traverse to the end of the list and then update its next pointer which takes O(n) time.

For iii, we just need to make head as head -> nex.

For iv, we first need to traverse to the second last node and update its next as null which takes O(n) time.

Q4: Give a Θ(n)-time nonrecursive procedure that reverses a singly linked list of n elements. The procedure should use no more than constant storage beyond that needed for the list itself.

Ans:

```
reverse(struct Node** head_ref) {
    struct Node* prev = NULL;
    struct Node* current = *head_ref;
    struct Node* next = NULL;
    while (current != NULL) {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    *head_ref = prev;
}
```

Q5: Which of the following sorting algorithms can be used to sort a

random linked list with minimum time complexity?

A. Insertion Sort

B. Quick Sort

C. Heap Sort

D. Merge Sort

Ans 17: D

Explanation: Both Merge sort and Insertion sort can be used for linked lists. The slow random-access performance of a linked list makes other algorithms (such as quicksort) perform poorly, and others (such as heapsort) completely impossible. Since the worst case time complexity of Merge Sort is O(nLogn) and Insertion sort is O(n2), merge sort is preferred.