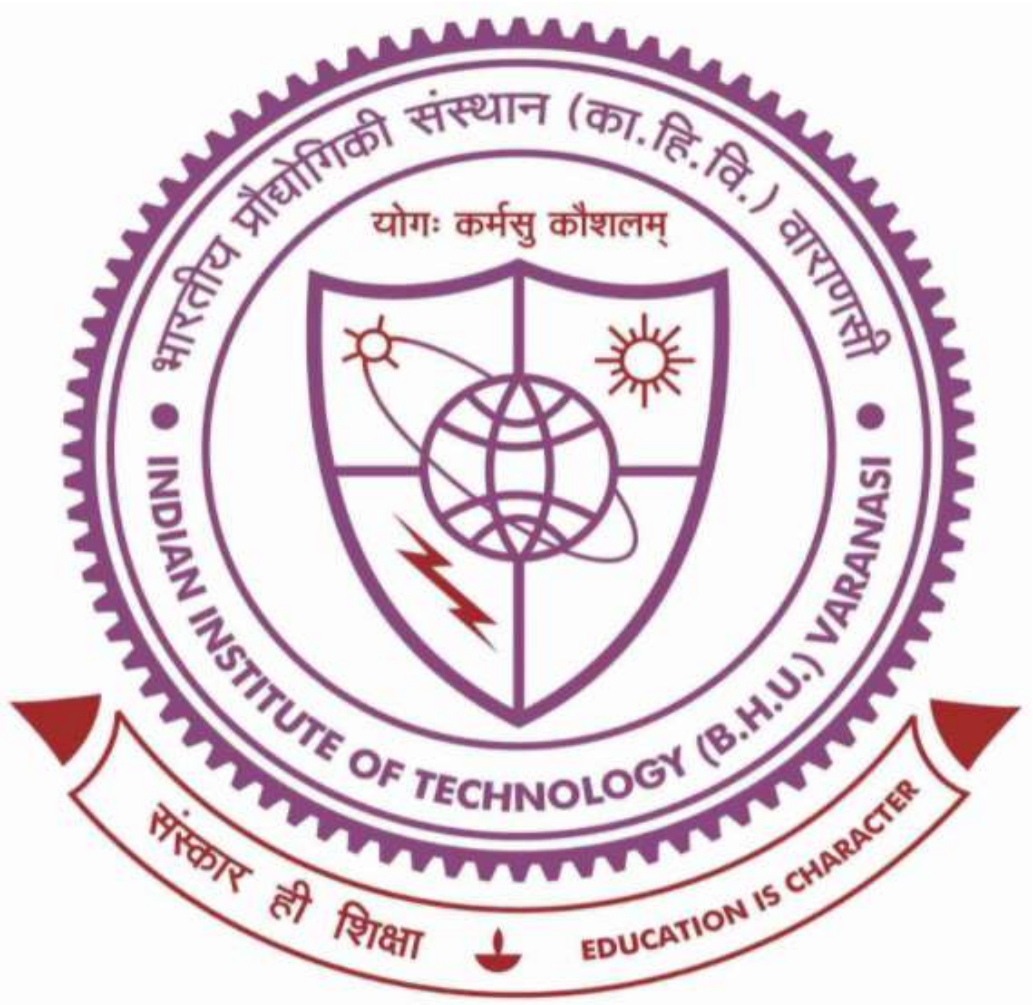


NAME: RUCHIRA NASKAR

ROLL NO.: 21075072

DEPARTMENT: COMPUTER SCIENCE AND ENGINEERING

SECTION: BA1

CSO 101 LAB ASSIGNMENT-9: POINTERS

EMAIL ID: ruchira.naskar.cse21@iitbhu.ac.in

SOLUTIONS

1. Match the following with reference to the following segment:

```
int x[3][5] = {
                { 1, 2, 3, 4, 5 },
                { 6, 7, 8, 9, 10 },
                { 11, 12, 13, 14, 15 }
            }, *n = &x ;
```

1.	*(*(x + 2) + 1)	a.	9
2.	*(*x + 2) + 5	b.	13
3.	*(*(x + 1))	c.	4
4.	*(*(x) + 2) + 1	d.	3
5.	* (*(x + 1) + 3)	e.	2
6.	*n	f.	12
7.	*(n +2)	g.	14
8.	(*(n + 3) + 1	h.	7
9.	*(n + 5)+1	i.	1
10.	++*n	j.	8
		k.	5
		l.	10
		m.	6

1. f) 12
2. j) 8
3. m)6
4. c)4
5. a)9
6. i)1
7. d)3
8. k)5
9. h)7
10. e)2

2. Write a C program to insert a substring into a string by using function and pointers.

CODE:

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
void insert_substring(char*, char*, int);
char* substring(char*, int, int);

int main()
{
   char str[100], substr[100];
   int position;

   printf("Enter main string:\n");
   scanf("%[^\n]s",str);
   fflush(stdin);

   printf("Enter a substring:\n");
   scanf("%[^\n]s",substr);
   fflush(stdin);

   printf("Enter the position to insert the substring in the main
string:\n");
   scanf("%d", &position);
   fflush(stdin);

   insert_substring(str, substr, position);

   printf("%s\n",str);

   return 0;
}

void insert_substring(char *a, char *b, int position)
{
   char *f, *e;
   int length;

   length = strlen(a);

   f = substring(a, 1, position - 1 );
   e = substring(a, position, length-position+1);

   strcpy(a, "");
   strcat(a, f);
```

```
   free(f);
   strcat(a, b);
   strcat(a, e);
   free(e);
}

char *substring(char *string, int position, int length)
{
   char *pointer;
   int c;

   pointer = malloc(length+1);

   if( pointer == NULL )
       exit(EXIT_FAILURE);

   for( c = 0 ; c < length ; c++ )
      *(pointer+c) = *((string+position-1)+c);

   *(pointer+c) = '\0';

   return pointer;
}
```

OUTPUT:

```
Enter main string:
Harry
Enter a substring:
07
Enter the position to insert the substring in the main string:
6
Harry07
```

3. Write a program using pointers to read in an array of integers and print its elements in reverse order.

CODE:

```c
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int tmp, n, i, j, *arr;
    printf("Enter size of array:");
    scanf("%d", &n);
    // allocates memory depending on size
    arr = calloc(sizeof(int), n);
    printf("Enter elements in array: ");
    for (i = 0; i < n; i++)
        scanf("%d", arr + i);

    printf("Before reversing the array: ");
    for (int i = 0; i < n; ++i)
        printf("%d ", *(arr + i));

    printf("\n");

    for (i = 0, j = n - 1; i < j; i++, j--)
    {
        // swap the elements
        tmp = *(arr + i);
        *(arr + i) = *(arr + j);
        *(arr + j) = tmp;
    }

    printf("After reversing the array: ");
    for (i = 0; i < n; i++)
        printf("%d ", *(arr + i));

    return 0;
}
```

OUTPUT:

```
Enter size of array:5
Enter elements in array: 45 54 34 21 321
Before reversing the array: 45 54 34 21 321
After reversing the array: 321 21 34 54 45
```

4. Using pointers, write a function that receives a character string and a character as argument and deletes all occurrences of this character in this string. The returned string should not have any holes.

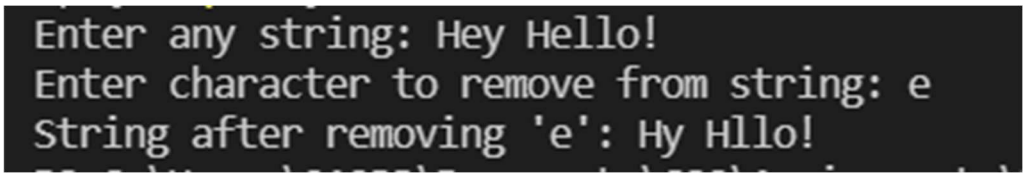
CODE:

```c
#include <stdio.h>
#include <string.h>
#define MAX_SIZE 100
void removeAll(char *, const char);
int main()
{
    char str[MAX_SIZE];
    char chr;
    printf("Enter any string: ");
    gets(str);
    printf("Enter character to remove from string: ");
    chr = getchar();
    removeAll(str, chr);
    printf("String after removing '%c': %s", chr, str);
    return 0;
}
void removeAll(char * str, const char toRemove)
{
    int i, j;
    int len = strlen(str);

    for(i=0; i<len; i++)
    {
        if(str[i] == toRemove)
        {
            for(j=i; j<len; j++)
            {
                str[j] = str[j+1];
            }

            len--;
            i--;
        }
    }
}
```

OUTPUT:


Enter any string: Hey Hello!
Enter character to remove from string: e
String after removing 'e': Hy Hllo!

5. Write a C program that demonstrates that uses the pointer increment operations to demonstrate the scale factor.

CODE:

```c
#include <stdio.h>
int main()
{
     int var, *ptr;
     ptr = &var;
     printf("Length of integer data type: %d\n",sizeof(int));
     printf("The address of ptr before increment: %u\n",ptr);
     ++ptr;
     printf("The address of ptr after increment: %u\n",ptr);
 }
```

OUTPUT:

```
Length of integer data type: 4
The address of ptr before increment: 6422296
The address of ptr after increment: 6422300
```

6. Write a C program that displays the addresses and values pointed by an array of integer pointers.

CODE:

```c
#include<stdio.h>
int main( )
{
        int n;
        printf("Enter array length: ");
        scanf("%d",&n);
        fflush(stdin);
        int arr[n];
        int i;
        printf("Enter the array 5 elements : ");
        for(i=0; i<5; i++)
        {
                scanf("%d", &arr[i]);
        }
        printf("\nArray elements with their addresses using pointers
: \n");

        for(i=0; i<5; i++)
        {
                printf("Value of arr[%d] = %d\t", i,*(arr+i));
                printf("Address of arr[%d] = %p\n",i,arr+i);
        }

        return 0;
}
```

OUTPUT:

```
Enter array length: 5
Enter the array 5 elements : 1 2 3 4 5

Array elements with their addresses using pointers :
Value of arr[0] = 1     Address of arr[0] = 0061FEDC
Value of arr[1] = 2     Address of arr[1] = 0061FEE0
Value of arr[2] = 3     Address of arr[2] = 0061FEE4
Value of arr[3] = 4     Address of arr[3] = 0061FEE8
Value of arr[4] = 5     Address of arr[4] = 0061FEEC
```

7. Write a C program that demonstrates the difference between pass by value and pass by reference.

In pass by value, the value of each argument (actual) in the call is copied into the corresponding formal arguments of the function which are called. Here, the changes made to the formal arguments in the called function have no effect on the values of actual arguments in the calling function. In pass by reference, the addresses of actual arguments in the call are copied into formal arguments of the called function. Using these addresses, we can have access to the actual arguments to manipulate them.

Code to demonstrate the difference between pass by value and pass by reference: -

```c
#include<stdio.h>
void swapv(int x,int y);
void swapr(int *, int *);
int main()
{
    //pass by value
    printf("pass by value\n");
    int a=10,b=20;
    swapv(a,b);
    printf("a = %d b = %d\n",a,b);
    printf("\n");
    //pass by reference
    printf("pass by reference\n");
    int p=10,q=20;
    swapr(&p,&q);
    printf("p = %d q = %d\n",p,q);
    printf("\n");
    return 0;
}
void swapv(int x, int y)
{
    int t;
    t=x;
    x=y;
    y=t;
    printf("x = %d y = %d\n",x,y);
}
void swapr(int *x, int *y)
{
    int t;
    t=*x;
    *x=*y;
    *y=t;
}
```

OUTPUT:

```
pass by value
x = 20 y = 10
a = 10 b = 20

pass by reference
p = 20 q = 10
```

8. Write a C program that checks whether two strings are equal by using pointers.

CODE:

```c
#include <stdio.h>
int stringsee(char *, char *);
int main()
{
    char str1[100];
    char str2[100];
    printf("Enter the first string : ");
    scanf("%s", str1);
    printf("\n");
    printf("Enter the second string : ");
    scanf("%s", str2);
    int check = stringsee(str1, str2); // calling function.
    if (check == 0)
        printf("strings are equal");
    else
        printf("strings are not equal");
    return 0;
}
int stringsee(char *a, char *b)
{
    int flag = 0;
    while (*a != '\0' && *b != '\0')
    {
        if (*a != *b)
        {
            flag = 1;
        }
        a++;
        b++;
    }
    if (flag == 0)
        return 0;
    else
        return 1;
}
```

OUTPUT:

```
Enter the first string : harryPotter

Enter the second string : harryPotter
strings are equal
```

```
Enter the first string : Good

Enter the second string : Okay
strings are not equal
```

9. Write a C program that demonstrates the difference between array of pointers and pointer to an array.

An array of pointers is a collection of addresses. The addresses present in an array can be those of isolated variables or array elements, etc. All rules that apply to an ordinary array apply to the array of pointers as well. Whereas, the entity pointer to an array is immensely useful when we need to pass a 2-D array to a function.

Code to demonstrate the difference between array of pointers and pointer to an array: -

```c
#include <stdio.h>
int main()
{
    // Array of pointers: all rules that apply to a normal array applies
here too
    int *arr[4]; // array of integer pointers
    int i = 20, j = 30, k = 40, l = 50;
    arr[0] = &i, arr[1] = &j, arr[2] = &k, arr[3] = &l;
    for (int i = 0; i < 4; i++)
        printf("%d\n", *(arr[i]));
    printf("\n");
    // Pointer to an array
    int s[4][2] = {
        {1234, 50}, {1212, 10}, {1434, 60}, {1312, 70}};
    int(*p)[2];
    int c, d, *pint;
    for (int c = 0; c < 4; c++)
    {
        p = &s[c];
        pint = (int *)p;
        printf("\n");
        for (int d = 0; d < 2; d++)
            printf("%d ", *(pint + d));
    }
    return 0;
}
```

OUTPUT:

```
20
30
40
50


1234 50
1212 10
1434 60
1312 70
```