USN | | | | | | | | | |

Internal Assessment Test 1 – December 2023

Sub:	**DSA**			Sub Code:	**BCS304**	Branch:	**AIDS & CSE(AIDS)**
Date:	**21/12/23**	Duration:	**90 minutes**	Max Marks: **50**	Sem/Sec:	**III -A, B & C**	**OBE**

Answer any FIVE FULL Questions — MARKS | CO | RBT

1 Illustrate KMP algorithm to find a string pattern with an example. [10] CO 1 L3
String: abcdabcabcabbabcbcabc
Pattern String:abcabbabc

Answer:

Knuth-Morris-Pratt (KMP) string matching algorithm runs in O(m+n) time to find all occurrences of pattern P in S. **In KMP algorithm, a preprocessing is done in pattern string P and an array of length m is calculated.**
The basic idea behind KMP's algorithm is: whenever we detect a mismatch (after some matches), we already know some of the characters in the text of the next window. We take advantage of this information to avoid matching the characters that we know will anyway match.
Unlike Brute-Force/Naïve algorithm, where we slide the pattern by one and compare all characters at each shift, we use a value from lps[] to decide the next characters to be matched. The idea is to not match a character that we know will anyway match.
It uses a preprocessed table called "Prefix Table" or "**Failure** Table" to skip characters comparison while matching.
How to use failure[] to decide next positions (or to know a number of characters to be skipped)?
We start comparison of pat[j] with j = 0 with characters of current window of text.
We keep matching characters str[i] and pat[j] and keep incrementing i and j while pat[j] and str[i] keep **matching**.
When we see a **mismatch**
We know that characters pat[0..j-1] match with str[i-j…i-1] (Note that j starts with 0 and increment it only when there is a match).
We also know (from above definition) that failure[j-1] is count of characters of pat[0…j-1] that are both proper prefix and suffix.
From above two points, we can conclude that we do not need to match these failure[j-1] characters with str[i-j…i-1] because we know that these characters will anyway match.

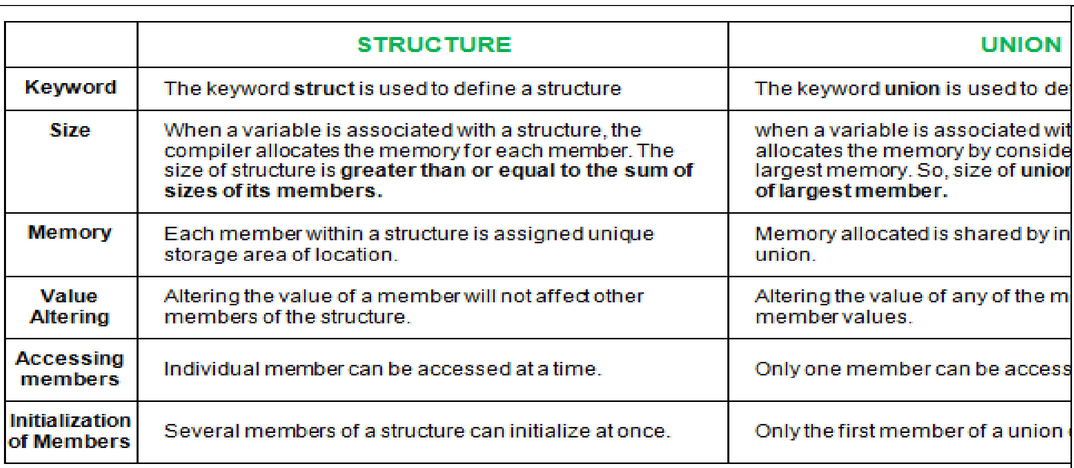
a	b	c	a	b	b	a	b	c
-1	-1	-1	0	1	-1	0	1	2

```
void fail(char *pat, int failure[])
{
        int i,j;
        int n = strlen(pat);
        failure[0] = -1;
        for (j=1; j<n; j++)

        {
                i = failure[j-1];
                while (( pat[j] != pat[i+1]) && (i>= 0))
                        i= failure[i];
                if (pat[j] == pat[i+1])
                        failure[j] = i+1;
                else
                        failure[j] = -1;
```

		} } abcdabcabcabbabcbcabc abcabbabc abcabbabc abcabbabc			
2		Convert the infix expression a/b – c+ d * e – a * c into postfix expression. Write a function to evaluate that postfix expression and trace that for given data a=6, b=3, c = 1, d = 2, e =4. Answer: Infix to postfix : ab/c-de*+ac*- (With tracing) POSTFIX EVALUATION : 63/1-24*+61*- = 3 (with tracing and function)	[10]	2	L3

3	a	What is structure? How it is different from array? Explain different types of structure declaration with examples and give difference between Union and Structure.
		
		Answer:
		
		Data are a collection of facts or simply values or sets of values.
		Data structure is representation of the logical or mathematical model of a particular organization of data.
		We can declare a structure using “struct” keyword. A structure must be declared first before using it just like all other data type. Structure can be declared by two ways.
		Tagged Declaration
		Typedef Declaration
		Typedef:
		typedef struct
		{
		data-type var-name1;
		data-type var-name2;
		:
		data-type var-nameN;
		}*identifier*; // global declaration
		Tagged:
		struct *tag_name*
		{
		data-type var-name1;
		data-type var-name2;
		:
		data-type var-nameN;
		};
			[5]	1	L2



	STRUCTURE	UNION
Keyword	The keyword **struct** is used to define a structure	The keyword **union** is used to de
Size	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 wit allocates the memory by conside largest memory. So, size of **unio of largest member.**
Memory	Each member within a structure is assigned unique storage area of location.	Memory allocated is shared by in union.
Value Altering	Altering the value of a member will not affect other members of the structure.	Altering the value of any of the m member values.
Accessing members	Individual member can be accessed at a time.	Only one member can be access
Initialization of Members	Several members of a structure can initialize at once.	Only the first member of a union

	b	Explain dynamic memory allocation functions in details Answer: Malloc Calloc Realloc Free With syntax and examples	[5]	1	L2
4		Explain how Polynomial addition is performed with algorithm? $P_1(x) = 4x^3+6x^2+7x+9$ $P_2(x) = 5x^4+8x^3+2x+3$ Answer: $P(x) = 5x^4+12x^3+6x^2+9x+13$ with algorithm/ function	[10]	1	L3
5		Explain Sparse matrices. Implement the fast transpose algorithm for it.	[10]	1	L3

$$\begin{bmatrix} 0 & 9 & 0 & 0 & 0 & 4 & 0 & 0 \\ 0 & 0 & 6 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 5 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 6 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Answer:

Sparse matrix:

5	8	8
0	1	9
0	5	4
1	2	6
1	6	1
2	3	5
2	6	1
3	6	3
4	2	6

Fast transpose:

5	8	8
1	0	9
2	1	6
2	4	6
3	2	5
5	0	4
6	1	1
6	2	1
6	3	3

```
void transpose(int trip1[][3],int trip2[][3])
{
   int x,y,z,n;
   trip2[0][0] = trip1[0][1];
   trip2[0][1] = trip1[0][0];
   trip2[0][2] = trip1[0][2];
   z=1;
   n=trip1[0][2];
   for(x=0;x<trip1[0][1];x++)
           for(y=1;y<=n;y++)
              /*if a column number of current triple==x
              then insert the current triple in b2 */
              if(x==trip1[y][1])
              {
                        trip2[z][0]=x;
                        trip2[z][1]=trip1[y][0];
                        trip2[z][2]=trip1[y][2];
                        z++;
              }
}
```

6		Define stack. Explain stacks operations using dynamic arrays. Implement the operations of stack using arrays	[10]	2	L3

Answer:

A stack is an ordered list in which the insertion (also called push and add) and deletion (also called pop and remove) are made at one end called the top.
Given a stack S=(a0, …, an-1), we say that a0 is the bottom element, an-1 is the top element and ai is on top of ai-1 for 0 < I < n.
Stack is also known as a Last-In-First-Out (LIFO) list.

```
int capacity=1; // capacity of stack. Initialized to 1
int top = -1;
int *stack;
int isEmpty()
{
      if (top < 0) return 1;
      else return 0;
}

int isFull()
{
      if (top >= capacity-1) return 1;
      else return 0;
}

void push(int item)
{
      if (top >= capacity -1)
      {
            // double the capacity and reallocate memory
            capacity = capacity * 2;
            stack=(int *) realloc(stack,
capacity*sizeof(int));
      }

      // increment top and then store the item.
      stack[++top] = item;
}
```

```
int pop()
{
      int item;
      if (top < 0)
            printf("Stack is Empty. Pop Failed\n");
      else
            // get the item to return and then decrement top.
            return stack[top--];
}

int main()
{
      int item, option=1;
      stack = (int *) malloc( sizeof (int)); // Allocate memory for
1 element
      while (option !=0)
      {
            printf("Enter option (1: push, 2: pop, 0:Exit):");
            scanf("%d", &option);
            switch (option)
            {
                  case 1: printf("Enter item to be pushed:");
                          scanf("%d", &item);
                          push(item);
                          break;
                  case 2: item = pop();
                          printf("Poped %d\n", item);
                          break;
                  case 0: printf("Exiting\n");
                          break;

                  default: printf("Invalid option. Retry\n");
            }
      }
}
```

CI **CCI** **HoD**

--All the Best--