

**Department of Computer Science and Engineering**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **USN** |  |  |  |  |  |  |  |  |  |  |

**THIRD SEMESTER TEST -I**

|  |  |
| --- | --- |
| Subject: Data Structures Using C | **Session**: July - Dec 2019 |
| **Subject Code:**18CSI301 | **Duration**: 90 Minutes |
| **Date of Examination:** /09/2019 | **Max Marks**: 03 X 15 = 45 |

**Note:**

* Answer **3 full questions** and each full question carries **15 Marks.**
* **Provide neat diagrams wherever applicable.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Q.No** | **Question** | **Marks** | **CO’s** | **Bloom’s**  **Level** |
| 1a | Illustrate why data structures is used. Explain the classification of data structure.  **Illustrate why data structures is used**  A data structure is a particular way of storing and organizing data in a computer’s memory so that it can be used efficiently. Data may be organized in many different ways; the logical or mathematical model of a particular organization of data is called a data structure. The choice of a particular data model depends on the two considerations first; it must be rich enough in structure to mirror the actual relationships of the data in the real world. On the other hand, the structure should be simple enough that one can effectively process the data whenever necessary.   * Data structure is important because it is used in almost every program or software system. * It helps to write efficient code, structures the code and solve problems. * Data can be maintained more easily by encouraging a better design or implementation. * Data structure is just a container for the data that is used to store, manipulate and arrange.   It can be processed by algorithms  **Explain the classification of data structure.**  Classification (Primitive and Non-Primitive) Primitive Data Type  * Primitive data types are the data types available in most of the programming languages. * These data types are used to represent single value. * It is a basic data type available in most of the programming language.  |  |  | | --- | --- | | **Data type** | **Description** | | Integer | Used to represent a number without decimal point. | | Float | Used to represent a number with decimal point. | | Character | Used to represent single character. | | Boolean | Used to represent logical values either true or false. |   **types of data structure** Non-Primitive Data Type  * Data type derived from primary data types are known as Non-Primitive data types. * Non-Primitive data types are used to store group of values. | 8  **2**  **6**  **3**  3 | CO1 | L1 |
| 1c | Write a C program with an appropriate structure definition and variable declaration to read and display information about an employee using nested structures. Consider the following fields: Ename, Eid, DOJ(Date, Month, Year) and Salary(Basic, DA, HRA). | 7 | CO1 | L3 |
| **OR** | | | | |
| 2a | With suitable examples, discuss the different dynamic memory allocation functions used in C.  **Dynamic memory allocation:**  It is important to develop algorithm based applications like **stacks and queues.**  **Static Memory or fixed size:**   * Declaring a primative variable, **int a;** * Memory allocated at runtime only * We call it as static means memory size is fixed   Ex: int arr(100)  Here we cannot store more than 100, so it is called fixed size. These are called static memory allocation.  **Dynamic memory allocation:**   * Size of the array may be increased or decreased based on the elements storing and deleting. * To allocate the memory dynamically predefined function is used [**Stdlib.h**]library * **Stdlib.h provides 4 important functions to allocate or deallocate functions.** * **Pointers conceot is used**  1. **Malloc:** used to allocate the memory to structures 2. **Calloc :** used to allocate the memory to arrays 3. **Realloc :** used to increase or decrease size of the array 4. **Free:** used to delete the memory 5. **Malloc:** used to allocate the memory to structures.   **Void ptr = malloc(Size\_type size);**  Void ptr: return type= generic pointer  Size\_t: argument= unsigned= positive integer value  Size: variable   * Whenever we allocate memory we should pass only positive integer value * On success : it returns base address of the memory block * Onfailure: it returns the null pointer   **Memory allocation using pointers**  Struct emp  {  Int eno  Char ename[20]  Float esal;  };  Void ptr=(struct emp\*)malloc(size of(struct epm));  If(ptr==null)  {  Printf(“out of memory error\n”);  }  Else  {  Printf(“enter employee details”);  Scanf(“%d,%s,%p; ptr🡪eno; ptr🡪 ename;ptr🡪esal”);  }  **Calloc()**   * Memory allocation to array * Void \*calloc(size\_t n, size\_t size) [‘n’ is array size, size:size of the element in array] * On success it returns address of the memory block * On failure ; it returns null pointer * Using calloc function we can allocate memory dynamically * We cannot increase or decrease the size of the array * So calloc is failed * Calloc taing the realloc function to increase or decrease the size of the array   **Realloc();**  Increase or decrease the size of the array  Void\* realloc(void \*ptr, size\_t size)  Generic pointer it can access any array size | 8  **2**  **2**  **2**  **2** | CO1 | L1 |
| 2b | Write C functions to perform the following string operations using pointers:   1. To concatenate two strings: (S1=”Data”; S2=”Structures”) 2. To reverse a string: S2 | 7 | CO1 | L3 |
|  |  |  |  |  |
| 3a | Discuss the disadvantages of linear queue. Illustrate the implementation of circular queue using **C** programming.  **Discuss the disadvantages of linear queue.**  Let, assume after insertion operations rear is shifted to last position in **queue**. It means, now **queue** is full. Now if a new element is inserted then overflow condition will occur. Now, if we delete some elements from **queue** then front will be increased by 1.  . **Illustrate the implementation of circular queue using C programming.**   * We know that when queue is implemented as an array, insertion is not possible after rear reaches the last position of array. * There may be vacant position in the array but they can’t be utilized. * To overcome limitation we use the concept of circular queue. * We can think of array to be logically circular, so that the two ends of the array wrap upto make a circle.   C:\Users\My\Downloads\WhatsApp Image 2019-09-02 at 9.13.17 AM.jpeg   * The insertion and deletion operation in a circular queue can be performed in a manner similar to that of queue but we have to take care of two things. * If value of rear is MAX-1, then instead of incrementing rear we will make it zero and then perform insertion. * Similarly when the value of front becomes MAX-1, it will not be incremented but will be reset to zero.   Let us see the various operations on a circular queue.  C:\Users\My\Downloads\New Doc 2019-09-11 09.35.43.jpg  C:\Users\My\Downloads\New Doc 2019-09-11 09.35.43_2.jpg  **}**  **Printf(“queue** elements :/n**”);**  **I=front;**  **If(front<=rear)**  **{**  **While(i=<=rear)**  **Printf(“%d”, cqueue\_arr[i++]);**  **}**  **Else**  **{**  **While(i<=MAX-1)**  **Printf(“%d”, cqueue\_arr[i++]);**  **I=0;**  **While()i<=rear)**  **Printf(“%d”, cqueue\_arr[i++]);**  **}**  **Printf(“/n”);**  **}**  **/\*end of display\*/** | 10  2  8 | CO3 | L3 |
| 3b | Mention the applications of stack.  **Stack** is used to check the proper opening and closing of parenthesis.  **Stack** is used to reverse a string. We push the characters of string one by one into **stack** and then pop character from **stack**. **Stack** is used to keep information about the active functions or subroutines.  C:\Users\My\Desktop\stack.png | 5 | CO2 | L2 |
| **OR** | | | | |
| 4a | Illustrate the concept of Tower of Hanoi for n=1, n= 2, n=3 where ‘n’ indicates the no of disks. Write program Tower of Hanoi using recursion.  **Illustrate the concept of Tower of Hanoi for n=1, n= 2, n=3 where ‘n’ indicates the no of disks.**   * The problem of tower of Hanoi is to move Disks from one pillar to another pillar using a temporary pillar. * We have a source pillar **A** which has a finate number of disks, * And these disks are placed on it in a decreasing order * i.e largest data is at the bottom and the smallest disk is at the top * now we want to place all these disks on destination pillar **C in the** same order * We can use a temporary pillar **B** to place the disk temporarly when ever required.   + We can move only disk from one pillar to another at a time   + Larger disk cannot be placed on smaller disk. * Suppose the number of disks on pillar **A is N.** * **First we** will solve the problem for n=1, n=2, n=3 and we will develop a general procedure for the solution. * Here **A**  is the source pillar * C is the destination pillar * B is the temporary pillar        * **These** were the solutions for n=1, n=2,n=3.from these solutions we can observe that first we move n-1 disks from source pillar (A) to temporary pillar B. * **And then move the largest nth** disk to th destination pillar (c). so the general solution for N disks can be written as  1. **Move upper n-1 disks from A to B using C**  as the temporary pillar. 2. **Move nth disk from A to C.** 3. **Move n-1 disks from BtoC using A**  as the temporary pillar.   **//Program to solve tower of Hanoi using recursion.**  **# include** <stdio.h>  Void tofh(int ndisk,chr source, char temp, char dest);  Main  {  Char source =”A”, temp=”B”, dest=”C”;  Int ndisk;  Printf(“enter the number of disk:”);  Scanf(“&d”, &ndisk);  Printf(“sequence is :/n”);  Tofh(ndisk,source,temp,dest);  }  Void tofh(int ndisk, char source, char temp, char dest)  {  If(ndisk==1)  {  Printf(“move disk %d from %c🡪%c/n, ndisk,source,dest”);  Return;  }  tofh(ndisk-1, source, dest, temp);  printf(“move disk &d from %c🡪%c/n, ndisk, source, dest”);  tofh(ndisk-1,temp,source,dest);  }/\*end of tofh\*/+ | 9  5  4 | CO3 | L3 |
| 4b | Explain in detail the pattern matching algorithms. Discuss the brute force pattern matching algorithm. Also solve it for the following pattern Pattern=”CAT” ; TEXT=”ABABNACATM”  **Explain in detail the pattern matching algorithms.**  **PATTERN MATCHING ALGORTHIM:**  **Pattern** matching algorithm is an act of checking a given sequence of token for the presence of the constituents of some pattern.   * Strings are sequence of characters * Strings are more often used than numbers * Some of important operations on the string are searching a word, find and replace operations etc. * String matching is most important problem * String matching consists of searching a query string or pattern.   EX: “P” in a given text “T”   * Generally size of the pattern to be searched is smaller than the given text * There are several applications for the string matching some of these are   Text editor, search engine’s, biological applications   * Since matching algorithms are used extensively, these should be efficient in terms of time and space. * Let P[1..M] is the pattern to be searched and its size is “M” * Assume that pattern occurs in “T” at position the output of the matching algorithm will be the integer I {where 1<=i<=n-m}if there are multiple occurrence of the pattern in the text, then some times.   **Let pattern “P”=CAT**  **TEXT=ABABNACATM**   * Then there is a match with the shift in the text “T”  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | | A | B | A | B | N | A | **C** | **A** | **T** | M |   PATTERN P= CAT  **BRUTE FORCE STRING MATCHING ALGORTHIM**  this is a simple and one in which we can compare a given pattern “p” with each of the surroundings of the text “T”, moving from left to right until match is found.   * Let Si is the substring of “T” beginning at the ith postion and where length is same as pattern “P”. * We compaer “P” character by character with the first substring s1 * If all the corresponding characters are same then the pattern “p” appears in ‘T’ at shift”1” * If the same of the character of s1 not matched with the corresponding characters of “p”,then we try for next substring s2 * This procedure continues till the input text exhausted * In this algorithm we have to compare “p” with n-m+1 substring of T | 6  3  3 | CO2 | L3 |
|  |  |  |  |  |
| 5a | Consider an array of elements arr [6]={23,78,45,8,32,56}, sort the following elements in ascending order using insertion sort. Write pseudo code / algorithm. | 8 | CO2 | L3 |
| 5b | With an algorithm evaluate the polish notation expression using stack.   1. 6 3 2 – 5 \* + 1 / 7 + 2. 5 3 2 + 8 \* + | 7 | CO3 | L3 |
| **OR** | | | | |
| 6a | Consider an array of elements {247,321,515,227,642,413,109,248,754,930} Arrange the given numbers using radix sort.. | 8 | CO2 | L3 |
| 6b | Represent the following sparse matrix using arrays and linked list. Also find its transpose.  15 0 0 22 0 -5  0 10 2 0 0 0  0 0 0 -4 0 0  0 0 0 0 0 0  91 0 0 0 0 0  0 0 28 0 0 0 | 7 | CO2 | L3 |

**Course Outcomes:**

|  |  |
| --- | --- |
|  | Understand the various types of data structures, operations and algorithms |
|  | Analyze the various algorithms used in linear and non-linear data structures. |
|  | Design the algorithm for stack, queues, list, trees and graphs. |
|  | Apply appropriate data structures for solving computing problems. |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| L1 | L2 | L3 | L4 | L5 | L6 |
| Remembering | Understanding | Applying | Analyzing | Evaluating | Creating |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **CO/PO: Mapping** | | | | | | | | | | | | |
| (H/M/L indicates strength of correlation) H-High, M-Medium, L-Low | | | | | | | | | | | | |
| **Course Outcome**  **(COs)** | **Program Outcome (POs)** | | | | | | | | | | | |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  | H | H | H | H | M | L | M | M | M | L | H | H |
|  | H | H | L | H | M | L | M | M | M | L | H | H |
|  | H | H | M | H | H | L | M | M | M | L | H | H |
|  | M | M | H | M | M | L | M | M | M | L | M | H |

**\*\*\*\*\***