1

UNIT

Array and Linked List

CONTENTS

PART-1

Introduction : Basic Terminology, Elementary Data OrganizationBuilt in Data Types in C.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 1.1. **Define data structure. Describe about its need and types.**

Why do we need a data type ?

AKTU 2014-15, Marks 05

Answer

Data structure :

1. A data structure is a way of organizing all data items that considers notonly the elements stored but also their relationship to each other.
2. Data structure is the representation of the logical relationship existing between individual elements of data.
3. Data structure is define as a mathematical or logical model of particular organization of data items.

Data structure is needed because :

1. It helps to understand the relationship of one element with the other.
2. It helps in the organization of all data items within the memory.

The data structures are divided into following categories :

1. **Linear data structure :**
   a. A linear data structure is a data structure whose elements form a sequence, and every element in the structure has a unique predecessor and unique successor.
   b. Examples of linear data structure are arrays, linked lists, stacksand queues.
2. **Non-linear data structure :**
   a. A non-linear data structure it is a data structure whose elements do notform a sequence. There is no unique predecessor or unique successor.
   b. Examples of non-linear data structures are trees and graphs.

Need of data type : The data type is needed because it determines what type of information can be stored in the field and how the data can be formatted.

Que 1.2. **Discuss some basic terminology used and elementarydata organization in data structures.**

Answer

Basic terminologies used in data structure :

1. **Data :** Data are simply values or sets of values. A data item refers to a single unit of values.
2. **Entity :** An entity is something that has certain attributes or properties which may be assigned values.
3. **Field :** A field is a single elementary unit of information representing an attribute of an entity.
4. **Record :** A record is the collection of field values of a given entity.
5. **File :** A file is the collection of records of the entities in a given entity set.

Data organization : Each record in a file may contain many field items, butthe value in a certain field may uniquely determine the record in the file.Such a field *K* is called a primary key, and the values k_1, k_2,... in such a field are called keys or key values.

Que 1.3. **Define data types. What are built in data types in C ?Explain.**

Answer

1. C data types are defined as the data storage format that a variable can store a data to perform a specific operation.
2. Data types are used to define a variable before to use in a program.
3. There are two types of built in data types in C :

a. **Primitive data types :** Primitive data types are classified as :

i. **Integer type :** Integers are used to store whole numbers.

Size and range of integer type on 16-bit machine :

Type	Size (bytes)	Range	Format specifier
int or signed int	2	– 32,768 to 32,767	%d
unsigned int	2	0 to 65,535	%u
short int or signed short int	1	– 128 to 127	%hd
unsigned short int	1	0 to 255	%hu
long int or signed long int	4	– 2,147,483,648 to 2,147,483,647	%ld
unsigned long int	4	0 to 4,294,967,295	%lu

ii. **Floating point type :** Floating types are used to store real numbers.

Size and range of floating point type on 16-bit machine :

Type	Size (bytes)	Range	Format specifier
Float	4	3.4E – 38 to 3.4E+38	%f
double	8	1.7E – 308 to 1.7E+308	%lf
long double	10	3.4E – 4932 to 1.1E+4932	%lf

iii. **Character type :** Character types are used to store characters value.

Size and range of character type on 16-bit machine :

Type	Size (bytes)	Range	Format specifier
char or signed char	1	– 128 to 127	%c
unsigned char	1	0 to 255	%c

iv. **Void type :** Void type is usually used to specify the type of functionswhich returns nothing.

b. **Non-primitive data types :**

i. These are more sophisticated data types. These are derived fromthe primitive data types.

ii. The non-primitive data types emphasize on structuring of a group of homogeneous (same type) or heterogeneous (different type) items. For example, arrays, lists and files.

PART-2

Algorithm, Efficiency of an Algorithm, Time and Space Complexity.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 1.4. Define algorithm. Explain the criteria an algorithm must satisfy. Also, give its characteristics.

Answer

1. An algorithm is a step-by-step finite sequence of instructions, to solve a well-defined computational problem.
2. Every algorithm must satisfy the following criteria :

   i. **Input :** There are zero or more quantities which are externally supplied.

ii. **Output :** At least one quantity is produced.

iii. **Definiteness :** Each instruction must be clear and unambiguous.

iv. **Finiteness :** If we trace out the instructions of an algorithm, then for all cases the algorithm will terminate after a finite number ofsteps.

v. **Effectiveness :** Every instruction must be basic and essential.

Characteristics of an algorithm :

1. It should be free from ambiguity.
2. It should be concise.
3. It should be efficient.

Que 1.5. How the efficiency of an algorithm can be checked ?Explain the different ways of analyzing algorithm.

Answer

The efficiency of an algorithm can be checked by :

1. Correctness of an algorithm
2. Implementation of an algorithm
3. Simplicity of an algorithm
4. Execution time and memory requirements of an algorithm

Different ways of analyzing an algorithm :

a. Worst case running time :

1. The behaviour of an algorithm with respect to the worst possiblecase of the input instance.
2. The worst case running time of an algorithm is an upper bound on the running time for any input.

b. Average case running time :

1. The expected behaviour when the input is randomly drawn from a given distribution.
2. The average case running time of an algorithm is an estimate ofthe running time for an “average” input.

c. Best case running time :

1. The behaviour of the algorithm when input is already in order. For example, in sorting, if elements are already sorted for a specific algorithm.
2. The best case running time rarely occurs in practice comparatively with the first and second case.

Que 1.6. Define complexity and its types.

Answer

1. The complexity of an algorithm *M* is the function *f*(*n*) which gives the running time and/or storage space requirement of the algorithm in terms of the size *n* of the input data.

2. The storage space required by an algorithm is simply a multiple of the data size *n*.
3. Following are various cases in complexity theory :
   - **a. Worst case :** The maximum value of *f*(*n*) for any possible input.
   - **b. Average case :** The expected value of *f*(*n*) for any possible input.
   - **c. Best case :** The minimum possible value of *f*(*n*) for any possibleinput.

Types of complexity :

1. **Space complexity :** The space complexity of an algorithm is the amountof memory it needs to run to completion.
2. **Time complexity :** The time complexity of an algorithm is the amount of time it needs to run to completion.

Que 1.7. What do you understand by complexity of an algorithm ? Compute the worst case complexity for the followingC code :

```
main()
{
int s = 0, i, j, n;
for (j = 0;  j < (3 * n); j++)
{
for (i = 0; i < n; i++)
{
s = s + i;
}
printf("%d", i);
}}
```

AKTU 2014-15, Marks 05

Answer

Complexity of an algorithm : Refer Q. 1.6, Page 1–6A, Unit-1.

Worst case complexity : $\Omega(n) + \Omega(3n) = \Omega(n)$

Que 1.8. How do you find the complexity of an algorithm ? What is the relation between the time and space complexities of analgorithm ? Justify your answer with an example.

AKTU 2015-16, Marks 10

Answer

Complexity of an algorithm : Refer Q. 1.6, Page 1–6A, Unit-1.

Relation between the time and space complexities of an algorithm :

1. The time and space complexities are not related to each other.
2. They are used to describe how much space/time our algorithm takes based on the input.
3. For example, when the algorithm has space complexity of :

a. O(1) *i.e.*, constant then the algorithm uses a fixed (small) amountof space which does not depend on the input. For every size of the input the algorithm will take the same (constant) amount of space.

b. $O(n)$, $O(n^2)$, $O(\log(n))$ - these indicate that we create additional objects based on the length of our input.

4. In contrast, the time complexity describes how much time our algorithm consumes based on the length of the input.
5. For example, when the algorithm has time complexity of :

a. O(1) *i.e.*, constant then no matter how big is the input it always takes a constant time.

b. $O(n)$, $O(n^2)$, $O(\log(n))$ - again it is based on the length of the input.

For example :

```
        function(list l) {                    function(list l) {
        for (node in l) {                     print("I got a list"); }
        print(node) ;
        }
    }
```

In this example, both take O(1) space as we do not create additional objects which shows that time and space complexity might be different.

PART-3

Asymptotic Notations : Big Oh, Big Theta, and Big Omega.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 1.9. **What is asymptotic notation ? Explain the big 'Oh' notation.**

Answer

1. Asymptotic notation is a shorthand way to describe running times for an algorithm.
2. It is a line that stays within bounds.
3. These are also referred to as 'best case' and 'worst case' scenarios respectively.

Big 'Oh' notation :

1. Big-Oh is formal method of expressing the upper bound of an algorithm's running time.
2. It is the measure of the longest amount of time it could possibly take forthe algorithm to complete.
3. More formally, for non-negative functions, $f(n)$ and $g(n)$, if there exists an integer n_0 and a constant $c > 0$ such that for all integers $n > n_0$.

$$f(n) \leq cg(n)$$

4. Then, $f(n)$ is Big-Oh of $g(n)$. This is denoted as : $f(n) \in O(g(n))$ *i.e.*, the set of functions which, as n gets large, grow faster than a constant time $f(n)$.

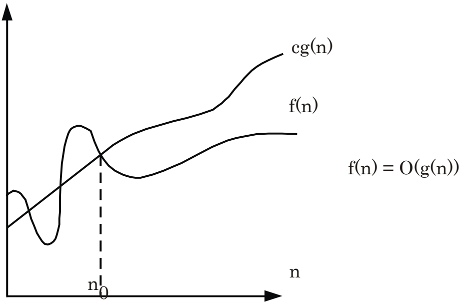


Fig. 1.9.1.

Que 1.10. What is complexity of an algorithm ? Explain various notations used to express the complexity of an algorithm.

OR

What are the various asymptotic notations ? Explain Big O notation.

AKTU 2017-18, Marks 07

Answer

Complexity of an algorithm : Refer Q. 1.6, Page 1–6A, Unit-1.

Notations used to express the complexity of an algorithm :

1. **θ-Notation (Same order) :**
   a. This notation bounds a function to within constant factors.
   b. We say $f(n) = \theta g(n)$ if there exist positive constants n_0, c_1 and c_2 such that to the right of n_0 the value of $f(n)$ always lies between $c_1 g(n)$ and $c_2 g(n)$ inclusive.

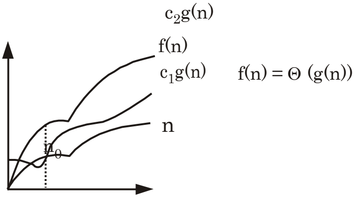


Fig. 1.10.1.

2. **Oh-Notation (Upper bound) :** Refer Q. 1.9, Page 1–8A, Unit-1.

3. **Ω-Notation (Lower bound) :**
   a. This notation gives a lower bound for a function to within a constant factor.

b. We write $f(n) = \Omega\ g(n))$ if there are positive constants n_0 and c suchthat to the right of n_0, the value of $f(n)$ always lies on or above $cg(n)$.

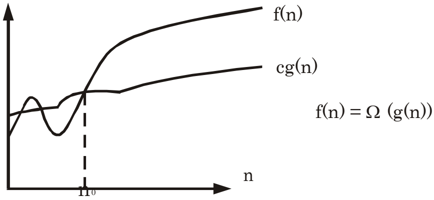


Fig. 1.10.2.

4. Little - Oh notation (o) :

a. It is used to denote an upper bound that is asymptotically tight because upper bound provided by O-notation is not tight.

b. We write $o(g(n)) = \{f(n)$: For any positive constant $c > 0$, if a constant $n_0 > 0$ such that $0 \leq f(n) < cg(n)\ \forall\ n \geq n_0\}$

5. Little omega notation (ω) :

a. It is used to denote lower bound that is asymptotically tight.

b. We write $\omega(g(n)) = \{f(n)$: For any positive constant $c > 0$, if a constant $n_0 > 0$ such that $0 \leq cg(n) < f(n)\}$

PART-4

Time-Space Trade-off, Abstract Data Types (ADT).

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 1.11. Explain time-space trade-off in brief with suitable example.

OR

What do you understand by time and space trade-off ? Define thevarious asymptotic notations. Derive the O-notation for linear search. **AKTU 2018-19, Marks 07**

Answer

Time-space trade-off :

1. The time-space trade-off refers to a choice between algorithmic solutionsof data processing problems that allows to decrease the running time of an algorithmic solution by increasing the space to store data and vice-versa.

2. Time-space trade-off is basically a situation where either space efficiency (memory utilization) can be achieved at the cost of time or time efficiency (performance efficiency) can be achieved at the cost of memory.

   For Example : Suppose, in a file, if data stored is not compressed, ittakes more space but access takes less time. Now if the data stored iscompressed the access takes more time because it takes time to run decompression algorithm.

Various asymptotic notation : Refer Q. 1.10, Page 1–9A, Unit-1.

Derivation :

Best case : In the best case, the desired element is present in the first position of the array, *i.e.*, only one comparison is made.

So, $T(n) = O(1)$.

Average case : Here we assume that ITEM does appear, and that is equallylikely to occur at any position in the array. Accordingly the number of comparisons can be any of the number 1, 2, 3, , n and each number occurs
with the probability $p = 1/n$. Then

$$T(n) = 1 \,.\, (1/n) + 2 \,.\, (1/n) + 3 \,.\, (1/n) \qquad + n \,.\, (1/n)$$
$$= (1 + 2 + 3 + \ldots\ldots + n) \,.\, (1/n)$$
$$= n \,.\, (n + 1)/2 \,.\, (1/n)$$
$$= (n + 1)/2$$

$= O((n + 1)/2)$ □ $O(n)$

Worst case : Worst case occurs when ITEM is the last element in the array or is not there at all. In this situation n comparison is made

So, $T(n) = O(n + 1)$ □ $O(n)$

Que 1.12. What do you understand by time-space trade-off ? Explain best, worst and average case analysis in this respect withan example. AKTU 2017-18, Marks 07

Answer

Time-space trade-off : Refer Q. 1.11, Page 1–10A, Unit-1.

Best, worst and average case analysis : Suppose we are implementing an algorithm that helps us to search for a record amongst a list of records. We can have the following three cases which relate to the relative success ouralgorithm can achieve with respect to time :

1. **Best case :**
   a. The record we are trying to search is the first record of the list.
   b. If $f(n)$ is the function which gives the running time and / or storage space requirement of the algorithm in terms of the size n of the input data, this particular case of the algorithm will produce a

complexity $C(n) = 1$ for our algorithm $f(n)$ as the algorithm will runonly 1 time until it finds the desired record.

2. **Worst case :**
   a. The record we are trying to search is the last record of the list.
   b. If $f(n)$ is the function which gives the running time and / or storage space requirement of the algorithm in terms of the size n of the input data, this particular case of the algorithm will produce a complexity $C(n) = n$ for our algorithm $f(n)$, as the algorithm will run$n$ times until it finds the desired record.

3. **Average case :**
   a. The record we are trying to search can be any record in the list.
   b. In this case, we do not know at which position it might be.
   c. Hence, we take an average of all the possible times our algorithm may run.
   d. Hence assuming for n data, we have a probability of finding anyone of them is $1/n$.
   e. Multiplying each of these with the number of times our algorithm might run for finding each of them and then taking a sum of all those multiples, we can obtain the complexity $C(n)$ for our algorithm$f(n)$ in case of an average case as following :

$$C(n) = 1 \cdot \frac{1}{2} + 2 \cdot \frac{1}{2} + \ldots + n \cdot \frac{1}{2}$$

$$C(n) = (1 + 2 + \ldots + n) \cdot \frac{1}{2}$$

$$C(n) = \frac{n(n+1)}{2} \cdot \frac{1}{n} = \frac{n+1}{2}$$

Hence in this way, we can find the complexity of an algorithm for average case as

$$C(n) = O((n + 1)/2)$$

Que 1.13. What do you mean by Abstract Data Type ?

Answer

1. An Abstract Data Type (ADT) is defined as a mathematical model of thedata objects that make up a data type as well as the functions that operate on these objects.
2. An Abstract Data Type (ADT) is the specification of the data type which specifies the logical and mathematical model of the data type.
3. It does not specify how data will be organized in memory and what algorithm will be used for implementing the operations.

4. An implementation chooses a data structure to represent the ADT.
5. The important step is the definition of ADT that involves mainly two parts :
   a. Description of the way in which components are related to each other.
   b. Statements of operations that can be performed on the data type.

PART-5

Array : Definition, Single and Multidimensional Array.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 1.14. Define array. How arrays can be declared ?

Answer

1. An array can be defined as the collection of the sequential memory locations, which can be referred to by a single name along with a numberknown as the index, to access a particular field or data.
2. The general form of declaration is :

   type variable-name [size];

   a. Type specifies the type of the elements that will be contained in thearray, such as int, float or char and the size indicates the maximum of elements that can be stored inside the array.
   b. For example, when we want to store 10 integer values, then we canuse the following declaration, int A[10].

Que 1.15. Write short note on types of an array.

Answer

There are two types of array :

1. **One-dimensional array :**
   a. An array that can be represented by only one-one dimension such as row or column and that holds finite number of same type of dataitems is called one-dimensional (linear) array.
   b. One dimensional array (or linear array) is a set of '*n*' finite numbers of homogeneous data elements such as :
      i. The elements of the array are referenced respectively by an index set consisting of '*n*' consecutive number.

ii. The elements of the array are stored respectively in successive memory locations.

'n' number of elements is called the length or size of an array. The elements of an array 'A' may be denoted in C language as :$A[0]$, $A[1]$, $A[2]$, ... $A[n-1]$

2. **Multidimensional arrays :**

a. An array can be of more than one dimension. There are no restrictions to the number of dimensions that we can have.

b. As the the dimensions increase the memory requirements increase drastically which can result in shortage of memory.

c. Hence a multidimensional array must be used with utmost care.

d. For example, the following declaration is used for 3-D array :int a [50] [50] [50];

PART-6

Representation of Arrays : Row Major Order, and ColumnMajor Order, Derivation of Index Formulae for 1–D, 2–D, 3–D and n–D Array.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 1.16. What is row major order ? Explain with an example.

AKTU 2014-15, Marks 05

Answer

1. In row major order, the element of an array is stored in computer memory as row-by-row.
2. Under row major representation, the first row of the array occupies the first set of memory locations reserved for the array, the second row occupies the next set, and so forth.
3. In row major order, elements of a two-dimensional array are orderedas :

A_{11}, A_{12}, A_{13}, A_{14}, A_{15}, A_{16}, A_{21}, A_{22}, A_{23}, A_{24}, A_{25}, A_{26}, A_{31}, ..., A_{46}, A_{51}, A_{52}, ..., A56

Example :

Let us consider the following two-dimensional array :

$$\begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \end{bmatrix}$$

a. Move the elements of the second row starting from the first element to the memory location adjacent to the last element of the first row.

b. When this step is applied to all the rows except for the first row, we have a single row of elements. This is the row major representation.

c. By application of above mentioned process, we get

$\{a, b, c, d, e, f, g, h, i, j, k, l\}$

Que 1.17. Explain column major order with an example.

Answer

1. In column major order the elements of an array is stored as column-by-column, it is called column major order.
2. Under column major representation, the first column of the array occupies the first set of memory locations reserved for the array, the second column occupies the next set, and so forth.
3. In column major order, elements are ordered as :

$A_{11}, A_{21}, A_{31}, A_{41}, A_{51}, A_{12}, A_{22}, A_{32}, A_{42}, A_{52}, A_{13}, \ldots, A_{55}, A_{16}, A_{26}, \ldots\ldots, A_{56}.$

Example : Consider the following two-dimensional array :

$$\begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \end{bmatrix}$$

a. Transpose the elements of the array. Then, the representation will be same as that of the row major representation.

b. Then perform the process of row-major representation.

c. By application of above mentioned process, we get

$\{a, e, i, b, f, j, c, g, k, d, h, l\}$.

Que 1.18. Write a short note on address calculation for 2D array.

OR

Determine addressing formula to find the location of $(i, j)^{th}$ elementof a $m \times n$ matrix stored in column major order.

OR

Derive the index formulae for 1-D and 2-D array.

Answer

1. Let us consider a two-dimensional array A of size $m \times n$. Like linear array system keeps track of the address of first element only, *i.e.*, base address of the array (Base (A)).
2. Using the base address, the computer computes the address of the element in the i^{th} row and j^{th} column *i.e.*, LOC ($A[i][j]$).

Formulae :

a. Column major order :

LOC($A[i][j]$) = Base (A) + $w[m(j$ – lower bound for column index)

$+ (i - \text{lower bound for row index})]$

$\text{LOC}(A[i][j]) = \text{Base }(A) + w[mj + i]$ in *C*/*C*++

b. Row major order :

$\text{LOC}(A[i][j]) = \text{Base }(A) + w[n(i - \text{lower bound for column index})$
$+ (j - \text{lower bound for row index})]$

$\text{LOC}(A[i][j]) = \text{Base }(A) + w[ni + j]$ in *C*/*C*++

where *w* denotes the number of words per memory location for the array *A* or the number of bytes per storage location for one element of the array.

Que 1.19. Explain the formulae for address calculation for 3-Darray with example.

Answer

In three-dimensional array, address is calculated using following two methods :

Row major order :

$\text{Location }(A[i, j, k]) = \text{Base }(A) + mn\,(k - 1) + n\,(i - 1) + (j - 1)$

Column major order :

$\text{Location }(A[i, j, k]) = \text{Base }(A) + mn\,(k - 1) + m\,(j - 1) + (i - 1)$

For example : Given an array [1..8, 1..5, 1..7] of integers. If Base (*A*) = 900 then address of element *A*[5, 3, 6], by using rows and columns methods are : The dimensions of *A* are : $M = 8$, $N = 5$, $R = 7$, $i = 5$, $j = 3$, $k = 6$

Row major order :

$$\begin{aligned} \text{Location }(A[i, j, k]) &= \text{Base }(A) + mn\,(k - 1) + n\,(i - 1) + (j - 1) \\ \text{Location }(A[5, 3, 6]) &= 900 + 8 \times 5(6 - 1) + 5(5 - 1) + (3 - 1) \\ &= 900 + 40 \times 5 + 5 \times 4 + 2 \\ &= 900 + 200 + 20 + 2 = 1122 \end{aligned}$$

Column major order :

$$\begin{aligned} \text{Location }(A[i, j, k]) &= \text{Base }(A) + mn\,(k - 1) + m\,(j - 1) + (i - 1) \\ \text{Location }(A[5, 3, 6]) &= 900 + 8 \times 5(6 - 1) + 8(3 - 1) + (5 - 1) \\ &= 900 + 40 \times 5 + 8 \times 2 + 4 \\ &= 900 + 200 + 16 + 4 = 1120 \end{aligned}$$

Que 1.20. Consider the linear arrays *AAA* [5 : 50], *BBB* [– 5 : 10] and *CCC* [1 : 8].

a. Find the number of elements in each array.

b. Suppose base (*AAA*) = 300 and *w* = 4 words per memory cell for *AAA*. Find the address of *AAA* [15], *AAA* [35] and *AAA* [55].

AKTU 2015-16, Marks 10

Answer

a. The number of elements is equal to the length; hence use the formula :

Length = UB – LB + 1

Length (AAA) = 50 – 5 + 1 = 46

Length (BBB) = 10 – (– 5) + 1 = 16

Length (CCC) = 8 – 1 + 1 = 8

b. Use the formula

LOC (AAA [i]) = Base (AAA) + w (i – LB)LOC

(AAA [15]) = 300 + 4 (15 – 5) = 340

LOC (AAA [35]) = 300 + 4 (35 – 5) = 420

AAA [55] is not an element of AAA, since 55 exceeds UB = 50.

Que 1.21. Suppose multidimensional arrays *P* and *Q* are declaredas *P*(– 2: 2, 2: 22) and *Q*(1: 8, – 5: 5, – 10 : 5) storedin column major order

i. Find the length of each dimension of *P* and *Q*.

ii. The number of elements in *P* and *Q*.

iii. Assuming base address (*Q*) = 400, *W* = 4, find the effective indices E_1, E_2, E_3 and address of the element *Q*[3, 3, 3].

AKTU 2018-19, Marks 07

Answer

i. The length of a dimension is obtained by Length = Upper Bound – Lower Bound + 1 Hence, the lengths of the dimension of P are,L_1 = 2 – (– 2) + 1 = 5; L_2 = 22 – 2 + 1 = 21 The lengths of the dimension of Q are,

L_1 = 8 – 1 + 1 = 8; L_2 = 5 – (– 5) + 1 = 11; L_3 = 5 – (– 10) + 1 = 16

ii. Number of elements in P = 21 × 5 = 105 elements Number of elements in Q = 8 × 11 × 16 = 1408 elements

iii. The effective index E_i is obtained from E_i = k_i – LB, where k_i is the givenindex and LB, is the Lower Bound. Hence,

E_1 = 3 – 1 = 2; E_2 = 3 – (– 5) = 8; E_3 = 3 – (– 10) = 13

The address depends on whether the programming language stores Q in row major order or column major order. Assuming Q is stored in column major order.

E_3L_2 = 13 × 11 = 143

E_3L_2 + E_2 = 143 + 8 = 151

(E_3L_2)L_1 = 151 * 8 = 1208

(E_2L_2+E_2)L_1 + E_1 = 1208 + 2 = 1210

Therefore, LOC(Q[3,3,3]) = 400 + 4(1210) = 400 + 4840 = 5240

PART-7

Application of Arrays, Sparse Matrices and their Representation.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 1.22. Write a short note on application of arrays.

Answer

1. Arrays are used to implement mathematical vectors and matrices, aswell as other kinds of rectangular tables. Many databases, small andlarge, consist of one-dimensional arrays whose elements are records.
2. Arrays are used to implement other data structures, such as lists, heaps, hash tables, queues and stacks.
3. Arrays are used to emulate in-program dynamic memory allocation, particularly memory pool allocation.
4. Arrays can be used to determine partial or complete control flow in programs, as a compact alternative to multiple "if" statements.
5. The array may contain subroutine pointers (or relative subroutine numbers that can be acted upon by SWITCH statements) that direct thepath of the execution.

Que 1.23. What are sparse matrices ? Explain.

Answer

1. Sparse matrices are the matrices in which most of the elements of the matrix have zero value.
2. Two general types of *n*-square sparse matrices, which occur in various applications, as shown in Fig. 1.23.1.
3. It is sometimes customary to omit block of zeros in a matrix as inFig. 1.23.1. The first matrix, where all entries above the main diagonal are zero or, equivalently, where non-zero entries can only occur on orbelow the main diagonal, is called a lower triangular matrix.
4. The second matrix, where non-zero entries can only occur on the diagonalor on elements immediately above or below the diagonal, is called tridiagonal matrix.

$$\begin{pmatrix} 4 & & & \\ 3 & -5 & & \\ 1 & 0 & 6 & \\ -7 & 8 & -1 & 3 \end{pmatrix} \qquad \begin{pmatrix} 5 & -3 & & & \\ 1 & 4 & 3 & & \\ & 9 & -3 & 6 & \\ & & 2 & 4 & -7 \end{pmatrix}$$

(a) Triangular matrix (b) Tridiagonal matrix

Fig. 1.23.1.

Que 1.24. Write a short note on representation of sparse matrices.

Answer

There are two ways of representing sparse matrices :

1. **Array representation :**
   i. In the array representation of a sparse matrix, only the non-zero elements are stored so that storage space can be reduced.
   ii. Each non-zero element in the sparse matrix is represented as (row, column, value).
   iii. For this a two-dimensional array containing three columns can be used. The first column is for storing the row numbers, the secondcolumn is for storing the column numbers and the third column represents the value corresponding to the non-zero element at (row, column) in the first two columns.
   iv. For example, consider the following sparse matrix :

$$\begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 4 & 3 & 0 \end{bmatrix}$$

The above matrix can be represented as :

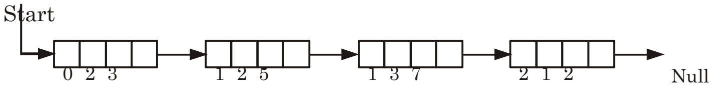
Row	Column	Value
0	0	2
1	1	1
2	1	4
2	2	3

2. **Linked representation :**
   i. In the linked list representation each node has four fields. Thesefour fields are defined as :
      a. **Row :** Index of row, where non-zero element is located.
      b. **Column :** Index of column, where non-zero element is located.
      c. **Value :** Value of non-zero element located at index – (row, column).
      d. **Next node :** Address of next node.

Node structure :

Row	Column	Value	Address

Example : $\begin{bmatrix} 0 & 0 & 3 & 0 \\ 0 & 0 & 5 & 7 \\ 0 & 2 & 0 & 0 \end{bmatrix}$



Que 1.25. Explain the upper triangular and lower triangular sparse matrices. Suggest a space efficient representation for sparse matrices.

Answer

1. The matrix, where all entries above the main diagonal are zero or equivalently, where non-zero entries can only occur, on or below the main diagonal, is called lower triangular matrix.
2. A matrix in which all the entries below the main diagonal are zero iscalled upper triangular matrix.

Space efficient representation for sparse matrices : Refer Q. 1.24,Page 1–19A, Unit-1.

PART-8

Linked List : Array Implementation and Pointer Implementation of Singly Linked List.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 1.26. Define the term linked list. Write a C program to implement singly linked list for the following function using array :

i. Insert at beginning **ii. Insert at end**
iii. Insert after element **iv. Delete at end**
v. Delete at beginning **vi. Delete after element**
vii. Display in reverse order

Answer

i. Linked list :

1. A linked list, or one-way list, is a linear collection of data elements, called nodes, where the linear order is given by means of pointers.

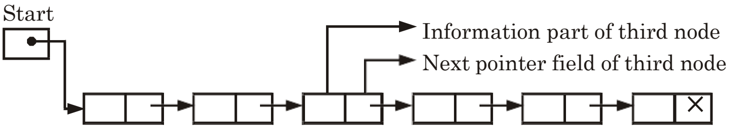

Start
Information part of third node
Next pointer field of third node

Fig. 1.26.1.

2. Each node is divided into two parts: the first part contains the information of the element, and the second part, called the link field or next pointer field, contains the address of the next node in the list.

Program :

```
#include<stdio.h>
#include<conio.h>
#include<alloc.h>
struct node {
      int info;
      struct node *link;
} ;
struct node *first;
void main( )
```

i. **Insert at beginning :** void

```
insert_beginning( ) {struct
node *ptr;
ptr = (struct node*)malloc(sizeof(struct node));if
(ptr == NULL) {
printf (“overflow\n” ) ;
return;
}
printf (“input new node information”);
scanf (“%d”, &ptr -> info) ;
ptr -> link = first;
first = ptr;
}
```

ii. **Insert at end :**

```
void insert_end( ) {
struct node *ptr; *cpt;
ptr = (struct node*)malloc(sizeof(struct node));if
(ptr == NULL) {
printf (“Link list is overflow\n”);
return;
}
printf (“input new node information”);
scanf (“%d”, &ptr -> info);
cpt = first;
while (cpt -> link != NULL)
cpt = cpt -> link;
cpt  -> link  = ptr; ptr
-> link = NULL;
```

iii. Insert after element :

```
void  insert_given_node(  ) {
struct  node  *ptr,  *cpt;
int data;
ptr  = (struct  node*)malloc(sizeof(struct  node));if
(ptr == NULL) {
printf  ("overflow\n");
return;
}
printf ("input new node information");
scanf ("%d", &ptr -> info);
printf ("input information of node after which insertion will be made") ; scanf
("%d", &data) ;
cpt = first;
while (cpt -> info != data)
cpt = cpt -> link;
ptr -> link = cpt -> link;
cpt -> link = ptr;
}
```

iv. Delete at end :

```
void delete_end( ) {
struct node *ptr,  *cpt; if
(first == NULL) { printf
("underflow\n");return;
}
ptr = first;
while (ptr -> link != NULL) {cpt
= ptr;
ptr = ptr -> link;
}
cpt -> link = NULL;
free (ptr);
}
```

v. Delete at beginning : void

```
delete_beginning( ) {struct
node *ptr;
if (first == NULL) { printf
("underflow\n")  ;return;
}
ptr = first;
first = ptr -> link;
free (ptr) ;
}
```

vi. Delete after element :

```
void
delete_given_info( ) {struct
node *ptr, *cpt;
int data;
if (first == NULL) { printf
("underflow\n" ) ;return;
}
ptr = first;
printf ("input information of node to be deleted") ;scanf
("%d", & data);
while  (ptr -> info != data) {
cpt = ptr;
ptr = ptr -> link;
}
cpt -> link = ptr -> link;
free  (ptr);
}
```

vii. Display in reverse order :

```
reverse_list( ) {
            ptr = First;
            cpt = NULL;
            while (ptr != NULL) {
                  cpt = ptr -> link;
                  ptr -> link = tpt;
                  cpt = ptr;
                  ptr = cpt;
            } }
```

Que 1.27. Write algorithm of following operation for linear linkedlist :

i. Traversal **ii. Insertion at beginning**
iii. Search an element **iv. Deletenodeatspecifiedlocation**
v. Deletion at end

Answer

i. Traversing a linked list : Let LIST be a linked list in memory. This algorithm traverses LIST, applying an operation PROCESS to each element of LIST. The variable PTR points to the node currently being processed.

```
1.  Set PTR := START [Initializes pointer PTR]
2.  Repeat Steps 3 and 4 while PTR != NULL
3.  Apply PROCESS to PTR -> INFO
4.  Set PTR := PTR -> LINK [PTR now points to the next node]
    [End of Step 2 loop]
5.  Exit
```

ii. Insertion at beginning : Here START is a pointer variable whichcontains the address of first node. ITEM is the value to be inserted.

```
1.  If (START == NULL) Then
2.  START = New Node              [Create a new node]
3.  START->INFO = ITEM            [Assign ITEM to INFO field]
4.  START->LINK = NULL            [Assign NULL to LINK field]
    Else
5.  Set PTR = START               [Initialize PTR with START]
6.  START = New Node              [Create a new node]
7.  START->INFO = ITEM            [Assign ITEM to INFO field]
8.  START->LINK = PTR             [Assign PTR to LINK field]
    [End of If]
9.  Exit
```

iii. Search an element : Here START is a pointer variable which contains the address of first node. ITEM is the value to be searched.

```
1.  Set PTR = START, LOC = 1           [Initialize PTR and LOC]
2.  Repeat While (PTR != NULL)
3.  If (ITEM == PTR -> INFO) Then    [Check if ITEM matches with
                                        INFO field]
4.  Print: ITEM is present at location LOC
5.  Return
6.  Else
7.  PTR = PTR -> LINK                  [Move PTR to next node]
8.  LOC = LOC + 1                      [Increment LOC]
9.  [End of If]
10. [End of While Loop]
11. Print: ITEM is not present in the list
12. Exit
```

iv. Delete node at specified position : Here START is a pointer variablewhich contains the address of first node. PTR is a pointer variable which contains address of node to be deleted. PREV is a pointer variable whichpoints to previous node. ITEM is the value to be deleted.

```
1.  If (START == NULL) Then            [Check whether list is empty]
2.  Print: Linked-List is empty.
3.  Else If (START -> INFO == ITEM) Then
                                       [Check if ITEM is in 1st node]
4.  PTR = START
5.  START = START -> LINK           [START now points to 2nd node]
6.  Delete PTR
7.  Else
8.  PTR = START, PREV = START
9.  Repeat While (PTR != NULL)
10. If (PTR -> INFO == ITEM) Then
                                   [If ITEM matches with PTR->INFO]
11. PREV -> LINK = PTR -> LINK [Assign LINK field of PTR to PREV]
12. Delete PTR
13. Else
```

```
14.  PREV = PTR                        [Assign PTR to PREV]
15.  PTR = PTR -> LINK                 [Move PTR to next node]
     [End of Step 10 If]
     [End of While Loop]
16.  Print: ITEM deleted
     [End of Step 1 If]
17.  Exit
```

v. Deletion at end : Here START is a pointer variable which contains theaddress of first node. PTR is a pointer variable which contains address of node to be deleted. PREV is a pointer variable which points to previous node. ITEM is the value to be deleted.

```
1.   If (START == NULL) Then           [Check whether list is empty]
2.   Print: Linked-List is empty.
3.   Else
4.   PTR = START, PREV = START
5.   Repeat While (PTR -> LINK != NULL)
6.   PREV = PTR                        [Assign PTR to PREV]
7.   PTR = PTR -> LINK                 [Move PTR to next node]
     [End of While Loop]
8.   ITEM = PTR -> INFO                [Assign INFO of last node to ITEM]
9.   If (START -> LINK == NULL) Then
                                              [If only one node is left]
10.  START = NULL                      [Assign NULL to START]
11.  Else
9.   PREV -> LINK = NULL
                         [Assign NULL to link field of second last node]
     [End of Step 9 If]
10.  Delete PTR
11.  Print : ITEM deleted
     [End of Step 1 If]
12.  Exit
```

Que 1.28. Implement linear linked list using pointer for followingfunctions :

i. Insert at beginning **ii. Insert at end**

iii. Insert after element **iv. Delete at end**

v. Delete at beginning **vi. Delete after element**

vii. Display in reverse order

Answer

```
#include<stdio.h>
#include<conio.h>
#include<process.h> typedef
struct simplelink {int data;
struct simplelink *next;
} node;
```

i. **Function to insert at beginning :**

```
node *insert_begin(node *p)
{
node *temp;
temp = (node *)malloc(sizeof(node));
printf("\nEnter the inserted data:");
scanf("%d",&temp->data);
temp->next = p;p
= temp;
return(p);
}
```

ii. **Function to insert at end :**node

```
*insert_end(node *p){ node
*temp, *q;
q = p;
temp=(node*)malloc(sizeof(node));
printf("\nEnter the inserted data;");
scanf("%d",&temp->data);
while(p->next != NULL)
{
p = p->next;
}
p->next = temp;
temp->next = (node *)NULL;return(q);
}
```

iii. **Function to insert after element:**

```
node *insert_after(node *p) {
node temp, *q;
int x;q
= p;
printf("\nEnter the data(after which you want to enter data):");
scanf("%d",&x);
while(p->data != x) {p
= p->next;
}
temp = (node *)malloc(sizeof(node));
printf("\nEnter the inserted data:");
scanf("%d",&temp->data);
temp->next = p->next;
p->next = temp; return
(q);
}
```

iv. **Function to delete last node :**

```
node *del end(node *p) {
```

```
node * q, *r;r
= p;
q = p;


if(p->next == NULL)
{
r = (node *)NULL;
}
else
{
while(p->next := NULL)
{
q = p;
p = p->next;
}
q->next = (node *)NULL;
}
free(p);
return(r);
}
```

v. Function to delete first node :node

```
*delete_begin(node *p) { node *q;
q = p;
q = p->next;
free(q);
return(p);
}
```

vi. Function to delete node after element :

```
node “delete_after(node, *p)
{
node *temp, *q;
int x;
q = p;
printf(“\nEnter the data(after which you want to delete):”);
scanf(“%d” ,&x);
while(p->data != x) {p
= p->next;
}
temp = p->next;
p->next = temp->next;
free(temp);
return (q);
}
```

vii. Function to reverse the list :

```
node *reverse(node *p) {
node *q, *r;
q = (node *)NULL;
while(p != NULL) {r
= q;
q = p;
p = p->next;
p->next = r;
}
return(q);
}
```

Que 1.29. What are the advantages and disadvantages of singlelinked list ?

Answer

Advantages :

1. Linked lists are dynamic data structures as it can grow or shrink duringthe execution of a program.
2. The size is not fixed.
3. Data can store non-continuous memory blocks.
4. Insertion and deletion of nodes are easier and efficient. Unlike array a linked list provides flexibility in inserting a node at any specified positionand a node can be deleted from any position in the linked list.
5. Many more complex applications can be easily carried out with linkedlists.

Disadvantages :

1. **More memory :** In the linked list, there is a special field called link fieldwhich holds address of the next node, so linked list requires extra space.
2. Accessing to arbitrary data item is complicated and time consuming task.

Que 1.30. Write an algorithm that reverses order of all theelements in a singly linked list.

Answer

1. To reverse a linear linked list, three pointer fields are used.
2. These are PREV, PTR, REV which hold the address of previous node, current node and will maintain the linked list.

Algorithm :

```
1. PTR = FIRST
2. TPT = NULL
3. Repeat step 4 while PTR != NULL
4. REV = PREV
```

```
5.    PREV = PTR
6.    PTR = PTR –> LINK

7.    PREV –> LINK = REV
      [End of while loop]
8.    START = PREV
9.    Exit
```

Que 1.31. **Write difference between array and linked list.**

AKTU 2014-15, Marks 05

Answer

S. No.	Array	Linked list
1.	An array is a list of finite number of elements of same data type *i.e.*, integer, real or string etc.	A linked list is a linear collectionof data elements called nodes which are connected by links.
2.	Elements can be accessed randomly.	Elements cannot be accessed randomly. It can be accessed only sequentially.
3.	Array is classified as : a. 1-D array b. 2-D array c. *n*-D array	A linked list can be linear, doubly or circular linked list.
4.	Each array element is independent and does not have a connection with previous element or with its location.	Location or address of element is stored in the link part of previous element or node.
5.	Array elements cannot be added, deleted once it is declared.	The nodes in the linked list can be added and deleted from the list.
6.	In array, elements can be modified easily by identifying the index value.	In linked list, modifying the node is a complex process.
7.	Pointer cannot be used in array.	Pointers are used in linked list.

PART-9

Time-Space Trade Off, Abstract Data Types (ADT).

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 1.32. Explain doubly linked list.

Answer

1. The doubly or two-way linked list uses double set of pointers, one pointingto the next node and the other pointing to the preceding node.
2. In doubly linked list, all nodes are linked together by multiple links which help in accessing both the successor and predecessor node for any arbitrary node within the list.
3. Every node in the doubly linked list has three fields :

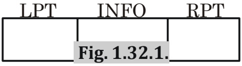


Fig. 1.32.1.

4. LPT will point to the node in the left side (or previous node) *i.e.*, LPT willhold the address of the previous node, RPT will point to the node in the right side (or next node) *i.e.*, RPT will hold the address of the next node.
5. INFO field store the information of the node.
6. A doubly linked list can be shown as follows :

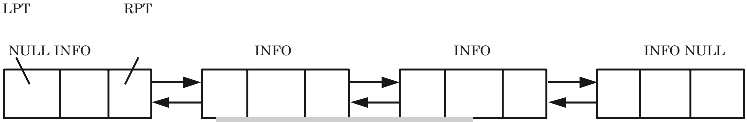


Fig. 1.32.2. Doubly linked list.

7. The structure defined for doubly linked list is:

```
struct node
      {
            int info;
            struct node *rpt;
            struct node *lpt;
      }     node;
```

Que 1.33. What are doubly linked lists ? Write C program to createdoubly linked list. **AKTU 2015-16, Marks 10**

Answer

Doubly linked list : Refer Q. 1.32, Page 1–30A, Unit-1.

Program :

```
# include<stdio.h>
# include<conio.h>#
include<alloc.h>
struct node
      {
      int info ;


      struct node *lpt ;
      struct node *rpt ;
      } ;
struct node *first ;
void main ( )
{
create ( ) ;
getch ( ) ;
}
void create ( )
{
struct node *ptr, *cpt ;
char ch ;
ptr = (struct node *) malloc (size of (struct node)) ;printf
("Input first node information") ;
scanf ("%d", & ptr → info) ;
ptr → lpt = NULL ;
first = ptr ;
do
{
cpt = (struct node *) malloc (size of (struct node)) ;printf
("Input next node information");
scanf ("%d", & cpt → info) ;
ptr → rpt = cpt ;
cpt → lpt = ptr ;
ptr = cpt ;
printf ("Press <Y/N> for more node") ;ch
= getch ( );
}
while (ch == 'Y') ; ptr
→ rpt = NULL ;
}
```

Que 1.34. Implement doubly linked list using pointer for followingfunctions :

i. Insert at beginning

ii. Insert at end

iii. Searching an element

iv. Delete at beginning

v. Delete at end

vi. Delete entire list

Answer

```
#include<stdio.h>
#include<conio.h>
typedef struct n{ int
data;
struct n *prev;
struct n *next;
}node;


node *head = NULL, *tail = NULL;
```

i. **Function to insert at beginning :**

```
void insert beg(node*h, int d) {node
*temp;
temp = (node *)malloc(sizeof(node));
temp->data = d;
temp->prev = NULL;
if(head == NULL)
{
temp->next = NULL;head
= tail = temp; return;
}
temp->next = h;
h->prev = temp;
h = h->prev;
head = h;
}
```

ii. **Function to insert at end :** void

```
insert_end(node *t, int d) {node
*temp;
temp = (node*)malloc(sizeof(node));
temp->data = d;
temp->next = NULL;
if(head == NULL) {
temp->prev = NULL;
head = tail = temp;
return;
}
temp->prev  =  t;
t->next = temp; t
= t->next;
tail = t;
}
```

iii. **Function to search an element :**

```
node *find(node *h, int aft) {
while(h->next != head && h->data != aft)
h = h->next;

if(h->next == head && h->data != aft)return
(node*) NULL;
else
return h;
}
```

iv. **Function to delete at beginning :**void

```
delete_beg(node *h, node *t) { if(head
== (node*)NULL) { printf("\nList is
empty.");
getch( );
return;
}
if(head == tail) {
free(h);
head = tail = (node *)NULL;return;
}
if(h->next == t) {
tail->prev = NULL;
head = tail;
}
else {
head = head->next;
head->prev = NULL;
}
free(h);
}
```

v. **Function to delete at end :**

```
void delete_end(node *h, node *t) {
if(head == (node *)NULL) {
printf("\nList is empty.");
getch( );
return;
}
if(head == tail) {
free(h);
head = tail = (node*)NULL;
return;
}
if(t->prev == h) {
head->next = NULL;
tail = head;
}
else {
tail = tail->prev;
tail->next = NULL;
}
free(t);
}


void display(node *h) {
while(h != NULL) {
printf(n"/%d", h->data);h
= h->next;
}
}
```

vi. Function to delete entire list :

```
void free_list(node *list) {
node *t;
while(list != NULL) {t
= list;
list = list->next;
free(t);
        }
        }
```

Que 1.35. Write algorithm of following operation for doubly linkedlist :

i. Traversal

ii. Insertion at beginning

iii. Delete node at specific location

iv. Deletion from end.

OR

Write an algorithm or C code to insert a node in doubly link list inbeginning.

AKTU 2014-15, Marks 05

Answer

i. Traversing of two-way linked list :

a. Forward Traversing :

```
1.  PTR ← FIRST.
2.  Repeat step 3 to 4 while PTR != NULL.
3.  Process INFO (PTR).
4.  PTR ← RPT (PTR).
5.  STOP.
```

b. Backward Traversing :

```
1.  PTR ← FIRST.
2.  Repeat step (3) while RPT (PTR) != NULL.
3.  PTR ← RPT (PTR)
4.  Repeat step (5) to (6) while PTR != NULL.
5.  Process INFO (PTR).
6.  PTR ←LPT (PTR).
7.  STOP.
```

ii. Insertion at beginning :

```
1.  IF PTR = NULL then Write OVERFLOWGo
    to Step 9
    [END OF IF]
2.  SET NEW_NODE = PTR
3.  SET PTR = PTR -> NEXT
4.  SET NEW_NODE -> DATA = VAL
5.  SET NEW_NODE -> PREV = NULL
6.  SET NEW_NODE -> NEXT = START
7.  SET HEAD -> PREV = NEW_NODE
8.  SET HEAD = NEW_NODE
9.  EXIT
```

iii. Delete node at specific location :

```
1.  IF HEAD = NULL then  Write UNDERFLOWGo
    to Step 9
    [END OF IF]
2.  SET TEMP = HEAD
3.  Repeat Step 4 while TEMP -> DATA != ITEM
4.  SET TEMP = TEMP -> NEXT
    [END OF LOOP]
5.  SET PTR = TEMP -> NEXT
6.  SET TEMP -> NEXT = PTR -> NEXT
7.  SET PTR -> NEXT -> PREV = TEMP
8.  FREE PTR
9.  EXIT
```

iv. Deletion from end :

```
1.  IF HEAD = NULL
    Write UNDERFLOW
    Go to Step 7
    [END OF IF]
2.  SET TEMP = HEAD
3.  Repeat Step 4 WHILE TEMP -> NEXT != NULL
4.  SET TEMP = TEMP -> NEXT
    [END OF LOOP]
5.  SET TEMP -> PREV -> NEXT = NULL
6.  FREE TEMP
7.  EXIT
```

Que 1.36. Write a program in C to delete a specific element in single linked list. Double linked list takes more space than singlelinked list for sorting one extra address. Under what condition, coulda double linked list more beneficial than single linked list.

AKTU 2018-19, Marks 07

Answer

Program to delete a specific element from a single linked list :

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

```
#include <stdlib.h>
// A linked list node
struct Node
{
int data;
struct Node *next;
};

/* Given a reference (pointer to pointer) to the head of a listand
an int, inserts a new node on the front of the list. */ void
push(struct Node** head_ref, int new_data)
{
struct Node* new_node = (struct Node*) malloc(sizeof(struct Node));
new_node->data = new_data;
new_node->next = (*head_ref);
(*head_ref) = new_node;
}
/* Given a reference (pointer to pointer) to the head of a listand
a position, deletes the node at the given position */
void deleteNode(struct Node **head_ref, int position)
{
// If linked list is empty if
(*head_ref == NULL)
return;
// Store head node
struct Node* temp = *head_ref;
// If head needs to be removedif
(position == 0)
{
*head_ref = temp->next; // Change head
free(temp); // free old head
return;
}
// Find previous node of the node to be deleted
for (int i = 0; temp != NULL && i < position – 1; i++)
temp = temp->next;
// If position is more than number of nodes if
(temp == NULL | | temp->next == NULL)
return;
// Node temp->next is the node to be deleted
// Store pointer to the next of node to be deletedstruct
Node *next = temp->next->next;
// Unlink the node from linked list
free(temp->next);  // Free memory
temp->next = next;  // Unlink the deleted node from list
}
// This function prints contents of linked list starting from
// the given node
```

```
void printList(struct Node *node)
{
while (node != NULL)
{
printf("%d ", node->data);
node = node->next;
}
}
/* Program to test above functions*/
int main()
{
/* Start with the empty list */
struct Node* head = NULL;
push(&head, 7);
push(&head,  1);
push(&head,  3);
push(&head,  2);
push(&head, 8); puts("Created
Linked List: ");printList(head);
deleteNode(&head, 4);
puts("\nLinked List after Deletion at position 4: ");printList(head);
return 0;
}
```

Double linked list is more beneficial than single linked list because :

1. A double linked list can be traversed in both forward and backward direction.
2. The delete operation in double linked list is more efficient if pointer tothe node to be deleted is given.
3. In double linked list, we can quickly insert a new node before a given node.
4. In double linked list, we can get the previous node using previous pointer but in singly liked list we traverse the list to get the previous node.

Circular Linked List.

PART-10

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 1.37. **What is meant by circular linked list ? Write the functions to perform the following operations in a doubly linkedlist.**

a. Creation of list of nodes.

b. Insertion after a specified node.

c. Delete the node at a given position.

d. Sort the list according to descending order

e. Display from the beginning to end.

AKTU 2016-17, Marks 15

Answer

Circular linked list : A circular list is a linear linked list, except that the lastelement points to the first element, Fig. 1.37.1 shows a circular linked listwith 4 nodes for non-empty circular linked list, there are no NULL pointers.

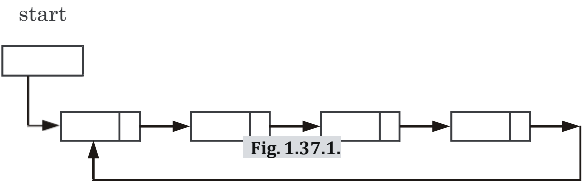


Fig. 1.37.1.

Functions :

a. To create a list : Refer Q. 1.33, Page 1–30A, Unit-1.

b. To insert after a specific node :

```
void insert_given_node ( )
{
struct node *ptr, *cpt, *tpt, *rpt, *lpt;int
m;
ptr = (struct node *) malloc (size of (struct node));if
(ptr == NULL)
{
printf (“OVERFLOW”);
return;
}
printf (“input new node information”);
scanf (“%d”, & ptr –> info);
printf (“input node information after which insertion”); scanf
(“%d”, & m);
cpt = first;
while (cpt –> info != m)
cpt = cpt –> rpt;
tpt = cpt –> rpt;cpt
–> rpt = ptr;
ptr –> lpt = cpt;
ptr –> rpt = tpt;
tpt –> lpt = ptr;

printf (“Insertion is done\n”);
}
```

c. To delete the node at a given position :

```
void deleteNode(int data) {
struct dllNode *nPtr, *tmp = head;if
(head == NULL) {
printf(“Data unavailable\n”);return;
} else if (tmp->data == data) {
nPtr = tmp->next;
tmp->next = NULL;
free(tmp);
head = nPtr;
totNodes– –;
} else {
while (tmp->next != NULL && tmp->data != data) {nPtr
= tmp;
tmp = tmp->next;
}
if (tmp->next == NULL && tmp->data != data) {
printf(“Given data unavailable in list\n”); return;
} else if (tmp->next != NULL && tmp->data == data) {
nPtr->next = tmp->next;
tmp->next->previous = tmp->previous;
tmp->next = NULL;
tmp->previous = NULL;
free(tmp);
printf(“Data deleted successfully\n”);
totNodes – –;
} else if (tmp->next == NULL && tmp->data == data) {
nPtr->next = NULL;
tmp->next = tmp->previous = NULL;free(tmp);
printf(“Data deleted successfully\n”);
totNodes– –;
}
}
}
```

d. To sort the list according to descending order :

```
void insertionSort() {
struct dllNode *nPtr1, *nPtr2;
int i, j, tmp;
nPtr1 = nPtr2 = head;
for (i = 0; i < totNodes; i++) {
tmp = nPtr1->data;
for (j = 0; j < i; j++) nPtr2
= nPtr2->next;
for (j = i; j > 0 && nPtr2->previous->data < tmp; j--) {
nPtr2->data = nPtr2->previous->data;
nPtr2 = nPtr2->previous;
}
nPtr2->data = tmp;
nPtr2 = head;
nPtr1 = nPtr1->next;
}
}
```

e. To display from the beginning to end :

```
void display()
{
if(head == NULL)
printf("\nList is Empty!!!");
else
{
struct Node *temp = head;
printf("\nList elements are: \n");
printf("NULL <--- ");
while(temp -> next != NULL)
{
printf("%d <===> ",temp -> data);
}
printf("%d ---> NULL", temp -> data);
}
}
```

Que 1.38. Write a C program to implement circular linked list forfollowing functions :

i. Searching of an element

ii. Insertion at specified position

iii. Deletion at the end

iv. Delete entire list

Answer

```
#include<stdio.h>
#include<conio.h>
typedef struct n{ int
data;
struct n *next;
}node;
node *head = NULL;
```

```
void insert_cir_end node *h, int d) {node
*temp;
temp = (node*)malloc(sizeof(node));
temp->data = d;
if(head == NULL) {head
= temp;
temp->next = head;
return;
}
while(h->next != head)h
= h->next;
temp->next = h->next;
h->next = temp;
}
```

i. Function to search an element :

```
node *find(node *h, int aft) {
while(h->next != head && h->data != aft)h =
h->next;
if(h->next == head && h->data != aft)return
(node*)NULL;
else
return h;
}
```

ii. Function to insert node at specified position :

```
void insert_cirsp_pos(node *h, int pos, int d)
{
node *temp, *loc;int
p = 0;
while(h->next != head && p < pos – 1)
{
      loc = h;
      p++;
      h = h->next;
}
if(pos > pos + 1 && h->next == head) || pos < 0)
{
printf("\nPosition does not exists.");
getch( );
}
if((p + 2) == pos) {loc
= h;
}
temp = (node*)malloc(sizeof(node));
temp->data = d;
temp->next = loc->next;
if(pos == 1) {
      h = head;
```

```
    while(h->next != head)h
    = h->next;
    h->next = temp;
    head = temp;
    }
else
    loc->next = temp;
}

void display (node *h) {
while (h->next != head) {
printf("%d", h->data);
h = h->next;
}
printf("%d", h->data);
}
```

iii. **Function to delete at the end :**void

```
delete_cir_end(node *h) { node
*temp;
if(head == NULL) {
printf("\nList is empty");getch(
);
}
if(h->next == head) {
printf("\nNode deleted. List is empty");getch(
);
head = NULL;
free(h); return;
}
while(h->next != head) {
temp = h;
h = h->next;
}
temp->next = h->next;
free(h);
}
```

iv. **Function to delete entire list :**

```
void free_list(node *list) {
node *t;
while(list != NULL) {t
= list;
list = list->next;
}
}
```

Que 1.39. Write an algorithm to insert a node at the end in a circular linked list.

AKTU 2017-18, Marks 07

Answer

```
1.  If PTR = NULL
2.  Write OVERFLOW
3.  Go to Step 1
    [END OF IF]
4.  SET NEW_NODE = PTR
5.  SET PTR = PTR -> NEXT
6.  SET NEW_NODE -> DATA = VAL
7.  SET NEW_NODE -> NEXT = HEAD
8.  SET TEMP = HEAD

9.  Repeat Step 10 while TEMP -> NEXT != HEAD
10. SET TEMP = TEMP -> NEXT
    [END OF LOOP]
11. SET TEMP -> NEXT = NEW_NODE
12. EXIT
```

PART-11

Operation on a Linked List, Insertion, Deletion, Traversal Polynomial Representation and Addition, Subtraction and Multiplications of Single Variable and Two Variable Polynomial.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 1.40. Write an algorithm to implement insertion, deletion and traversal on a singly linked list.

Answer

Refer Q. 1.27, Page 1–23A, Unit-1.

Que 1.41. Write a C program to implement insertion, deletionoperation on a doubly linked list.

Answer

Refer Q. 1.34, Page 1–31A, Unit-1.

Que 1.42. Write a C function for traversal operation on a doubly linked list.

Answer

Function for forward traversing :

```
void ftraverse ( )
{
      struct node *ptr;
      printf (“forward traversing :/n”);ptr
      =first ;
      while (ptr != NULL)
            {
            printf (“%d \n”, ptr -” info) ;ptr
                  = ptr -> rpt;
            }
      }
```

Function for backward traversing :

```
void btraverse ( )
      {
            struct node *  ptr ;
            printf (“Backward traversing :\n”)ptr
            = first ;
            while (ptr → rpt != NULL)ptr
                  = ptr → rpt ;
            while (ptr != NULL)
            {
                  printf (“%d \n”, ptr -> info) ;ptr
                  = ptr -> lpt;
            }
      }
```

Que 1.43. Write a program in C to implement insertion, deletion and traversal in circular linked list.

Answer

```
#include<stdio.h>
#include<conio.h>
#include<alloc.h>
struct node
      {
            int info;
            struct node  *link;
      } ;
      struct node *first;
      void main( )
```

```
    {

    void create( ), traverse( ), insert_beg( ), insert_end( ),
    delete_beg( ), delete_end( );
    clrscr( ) :
    create( ) ;
    traverse( ) ;
    insert_beg( ) ;
    traverse( ) ;
    insert_end( ) ;
    traverse( ) ;
    delete_beg( ) ;
    traverse( ) ;
    getch( ) ;
}
void create( )
{
    struct node *ptr, *cpt ;char
    ch ;
    ptr = (struct node *) malloc (size of (struct node)) ;printf(“input
    first node”) ;
    scanf(“%d” & ptr -> info) ;
    first = ptr ;
    do {
    cpt = (struct node *) malloc (size of (struct node)) ;
    printf (“Input next node”) ;
    scanf (“%d” & cpt -> info) ;
    ptr -> link = cpt ;
    ptr = cpt ;
    print f (“Press <Y/N> for more node”) ;ch
    = getch ( ) ;
}
while (ch == “Y”);ptr
-> link = first ;
}
void traverse ( )
{
    struct node *ptr ;
    printf (“Traversing of link list ; \n”) ;ptr
    = first ;
    while = (ptr != first)
{
    printf (“%d\n”, ptr -> info) :ptr
    = ptr -> link ;
}
}
void   insert_beg   (   )
{
```

```
struct node  *ptr;


ptr = (struct node*) malloc (sizeof (struct node));if
(ptr == NULL)
{     printf ("overflow\n" ) ;
      return ;
}
printf ("Input New Node");
scanf ("%d", &ptr -> info) ; cpt
= first ;
while (cpt -> link != first)
      {
      cpt = cpt -> link ;
      }
ptr -> Link = first;
first = ptr ;
cpt -> link = first ;
}
void insert_end( )
{
struct node *ptr; *cpt;
ptr = (struct node*) malloc (sizeof (struct node));if
(ptr == NULL)
{
printf("overflow\n") ;
return ;
}
printf ("Input New Node information");scanf
("%d", &ptr -> info);
cpt = first;
while (cpt -> link != first) ;cpt
= cpt -> link;
cpt -> link = ptr; ptr
-> link = first ;
}
void delete_beg ( )
{
struct node *ptr, *cpt ;if
(first == NULL)
{
printf ("underflow\n") ;
return;
}
cpt = first;
while (cpt -> link != First)cpt
      = cpt -> link ;
first = ptr -> link;
cpt -> link = first ;
```

```
free (ptr) ;
}
void delete_end( )
{


struct node *ptr, *cpt;if
(first == NULL)
{
        printf (“underflow\n”);
        return;
}
cpt = first;
while (cpt -> link != first)
{
ptr = cpt;
cpt = cpt -> link;
}
ptr -> link = first;
free (cpt);
        }
```

Que 1.44. Explain the method to represent the polynomialequation using linked list.

Answer

1. In the linked representation of polynomials, each node should consist of three elements, namely coefficient, exponent and a link to the next term.
2. The coefficient field holds the value of the coefficient of a term, the exponent field contains the exponent value of that term and the linkfield contains the address of the next term in the polynomial.

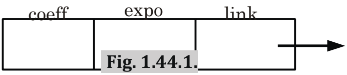


Fig. 1.44.1.

For example : Let us consider the polynomial of degree 4 *i.e.*, $3x^4 + 8x^2 + 6x + 8$ can be written as 3 * power (x, 4) + 8 * power (x, 2) + 6 * power(x, 1) + 8 * power (x, 0)

It can be represented as linked list as

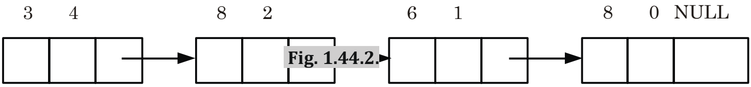


Fig. 1.44.2.

3. The link coming out of the last node is NULL pointer.

   In case of polynomial of 3 variables *i.e.*, x, y, z can also be represented as linked list as shown in Fig. 1.44.3.

power x	power y	power z	coeff	next

Fig. 1.44.3.

For example : Let us consider the following polynomial of 3 variable$3x^2 + 2xy^2 + 5y^3 + 7yz$.

We can replace each term of the polynomial with node of the linked list as

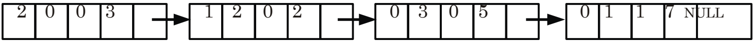


Fig. 1.44.4.

Que 1.45. Explain the method to represent the polynomial equation using linked list. Write and explain method to add twopolynomial equations using linked list.

Answer

Representation of polynomial : Refer Q. 1.44, Page 1–47A, Unit-1.

Addition of two polynomials using linked lists :

Let p and q be the two polynomials represented by the linked list.

1. While p and q are not null, repeat step 2.
2. If powers of the two terms are equal then,

   if the terms do not cancel then insert the sum of the terms into the sum (resultant)

   Polynomial

   Update p

   Update q

   Else if the (power of the first polynomial) > (power of second polynomial)

   Then insert the term from first polynomial into sum polynomial Update p

   Else insert the term from second polynomial into sum polynomialUpdate q
3. Copy the remaining terms from the non-empty polynomial into the sum polynomial.

Example : Let us consider the addition of two polynomials of single variable $5x^4 + 6x^3 + 2x^2 + 10x + 4$ and $7x^3 + 3x^2 + x + 7$. We can visualize this as follows :

$$\begin{array}{r} 5x^4 + 6x^3 + 2x^2 + 10x + 4 \\ + 7x^3 + 3x^2 + x + 7 \\ \hline 5x^4 + 13x^3 + 5x^2 + 11x + 11 \end{array}$$

i.e., to add two polynomials, compare their corresponding terms startingfrom the first node and move towards the end node.

Que 1.46. **Write and explain method to multiply polynomial equation using linked list.**

Answer

1. The multiplication of polynomials is performed by multiplying coefficient and adding the respective power.
2. To produce the multiplication of two polynomials following steps are performed :
   a. Check whether two given polynomials are non-empty. If anyone polynomial is empty then polynomial multiplication is not possible.So exit.
   b. Second polynomial is scanned from left to right.
   c. For each term of the second polynomial, the first polynomial is scanned from left to right and its each term is multiplied by the term of the second polynomial, *i.e.*, find the coefficient by multiplying the coefficients and find the exponent by adding the exponents.
   d. If the product term already exists in the resulting polynomial then its coefficients are added, otherwise a new node is inserted to represent this product term.

   For example : Let us consider two polynomial $8x^4 + 6x^2 + 5x + 2$ and$3x^2 + x + 2$ and perform multiplication as

$$
\begin{array}{l}
\qquad\qquad 8x^4 + 6x^2 + 5x + 2 \\
\qquad\qquad\qquad \times\ 3x^2 + x + 2 \\
\hline
24x^6 \qquad + 18x^4 + 15x^3 + 6x^2 \\
\quad + 8x^5 \qquad + 6x^3 + 5x^2 + 2x \\
\qquad\quad +16x^4 \qquad + 12x^2 + 10x + 4 \\
24x^6 + 8x^5 + 34x^4 + 21x^3 + 23x^2 + 12x + 4 \\
\hline
\end{array}
$$

VERY IMPORTANT QUESTIONS

Following questions are very important. These questionsmay be asked in your SESSIONALS as well as UNIVERSITY EXAMINATION.

Q. 1. Define data structure. Describe about its need and types.Why do we need a data type ?

Ans. Refer Q. 1.1.

Q. 2. What do you understand by complexity of an algorithm ? Compute the worst case complexity for the following C code :

```
main()
{
int s = 0, i, j, n;
for (j = 0;  j < (3 * n); j++)
{
for (i = 0; i < n; i++)
{
s = s + i;
}
printf("%d", i);
}}
```

Ans. Refer Q. 1.7.

Q. 3. How do you find the complexity of an algorithm ? What is the relation between the time and space complexities of an algorithm ? Justify your answer with an example.

Ans. Refer Q. 1.8.

Q. 4. What are the various asymptotic notations ? Explain Big O notation.

Ans. Refer Q. 1.10.

Q. 5. What do you understand by time and space trade-off ? Definethe various asymptotic notations. Derive the O-notation for linear search.

Ans. Refer Q. 1.11.

Q. 6. What do you understand by time-space trade-off ? Explain best, worst and average case analysis in this respect withan example.

Ans. Refer Q. 1.12.

Q. 7. Suppose multidimensional arrays *P* and *Q* are declared as $P(-2:2, 2:22)$ and $Q(1:8, -5:5, -10:5)$ stored in column majororder

i. Find the length of each dimension of *P* and *Q*.

ii. The number of elements in *P* and *Q*.

iii. Assuming base address $(Q) = 400$, $W = 4$, find the effectiveindices E_1, E_2, E_3 and address of the element $Q[3, 3, 3]$.

Ans. Refer Q. 1.21.

Q. 8. Write difference between array and linked list.

Ans. Ref