

SCHOOL OF COMPUTING AND INFORMATION TECHNOLOGY

**Data Structures Lab**

For

**Third Semester**

B. Tech in CSE (AIML), ISE, SE and CSIT

**INDEX**

|  |  |  |
| --- | --- | --- |
| **SL. No** | **Contents** | **Page. no** |
| 1 | Lab Objectives | 3 |
| 2 | Lab Outcomes | 3 |
| 3 | Lab Requirements | 4 |
| 4 | Guidelines to Students | 5 |
| 5 | List of Lab Exercises | 6 |
| 6 | Solutions for Lab Exercises | 8 |
| 7 | List of Lab Assignments | 65 |
| 8 | List of Viva Questions | 66 |
|  | Learning Resources and References | - |

**1. Lab Objectives:**

The objectives of this course are to:

1. Provide hands on skills on implementing different data-structures.
2. Analyze and Implement different applications of data-structures.
3. Identify and illustrate various applications of different data-structures.
4. Generate softwares as per user requirements
5. Develop industry standard data-structure applications for various domains.

**2. Lab Outcomes:**

On successful completion of this course; student shall be able to:

1. Design and implement a different data-structures.
2. Use appropriate data-structure for solving different problems.
3. Design and build a GUI application using a front end tool.
4. Design and develop applications based on different data structures like string reversal, sorting, searching, creating databases etc.

|  |
| --- |
| **3.Lab Requirements:** |
| Following are the required hardware and software for this lab, which is available in the laboratory.   * **Hardware:** Desktop system or Virtual machine in a cloud with OS installed. Presently in the Lab, Pentium IV Processor having 1 GB RAM and 250 GB Hard Disk is available. Desktop systems are dual boot having Windows as well as Linux OS installed on them. * **Software:** C-compiler. Many C-compilers are available. As we have dual boot, we have *gcc* compiler in Linux environment (fedora 8 to fedora 20), and Turbo C-IDE on Windows Environment. |
| This data structure programming manual is designed to increase the practical knowledge, thinking ability and creativity of students. Here, students are expected to read, think, design and implement the programs for given problems in ”C”. The Linux and windows environments are used as explained below.  **Linux environment:** All the programs have been written using “C” and tested using the *cc* compiler in Linux environment.  Steps for the successful program completion are :   1. Open vi editor using the shell command   *$vi <filename.c>*   1. Write the C-code according to the program logic. Save the program and exit (Esc + :wq) to get shell command prompt. 2. Compile the program using gcc compiler using the following shell command   *$cc <filename.c>*  Where, *cc* is the C compiler and *filename.c* is the input file name.   1. Execute the program using the following shell command   *$./exeFileName*  Where, *exeFileName* is the input file name.  **Windows environment:** Programs can be edited, compiled and executed using Turbo C/C++ IDE. All operations like opening a file, saving a file, compiling the program and executing the program are listed under different menus inside the IDE. Keyboard shortcuts are also available for the above mentioned operations.  **Note:** Every problem in the Lab manual includes problem statement, learning outcomes, theoretical description, algorithm, program, program description, expected results, implementation phase, simulation of syntax and logical errors, final program with results.  **Recommendation**: It is advised to use Linux environment for executing the programs given as part of this manual. |

## Guidelines to Students

* + Equipment in the lab for the use of student community. Students need to maintain a proper decorum in the computer lab. Students must use the equipment with care. Any damage is caused is punishable.
  + Students are required to carry their observation / programs book with completed exercises while entering the lab.
  + Students are supposed to occupy the machines allotted to them and are not supposed to talk or make noise in the lab. The allocation is put up on the lab notice board.
  + Lab can be used in free time / lunch hours by the students who need to use the systems should take prior permission from the lab in-charge.
  + Lab records need to be submitted on or before date of submission.
  + Students are not supposed to use flash drives.

**List of Lab Exercises**

|  |  |
| --- | --- |
| **Sl.**  **No.** | **Name of the Lab Program** |
| 1. | Consider the Software company “XYZ” with 100 employees. Design and develop a C program to construct an array to store the id of 100 employees and use different searching techniques to search for an employee with particular employee id. |
| 2. | Polynomials are used to graph real world curves. For example, roller coaster designers may use polynomials to describe the curves in their rides. Combinations of polynomial functions are sometimes used in economics to do cost analysis. Design and develop a C program to add two polynomials of any degree using single dimensional array. |
| 3. | Real-life quantities that are naturally described by complex numbers rather than real numbers. Design and develop a C program to add and subtract ‘n’ complex numbers using structure. |
| 4. | Reversal of string is used to check for palindrome or not. Design and Develop a C program to reverse a string using stack. |
| 5. | The compilers always convert infix expression into postfix to perform further operations like parsing, lexical analysis etc. Design and Develop a C program to convert an infix expression into postfix using stack. |
| 6. | Evaluation of postfix expressions is done by compilers during the compilation process. Design and Develop a C program to evaluate a postfix expression using stack. |
| 7. | When jobs are entered in system they are stored in the job queue and whenever the processor is free the jobs are moved to ready queue for execution. Consider a college is having a printer. Design and develop a C program to implement job scheduling in the printer.  **Hint:** Store the jobs based on their job id in queue. |
| 8. | In a machine gun bullets are always inserted in circular queue. Design and develop a C program to process bullets in machine gun.  **Hint:** Bullet insertion and firing (deletion) can be implemented using circular queue. Bullets are given with specific id. |

|  |  |
| --- | --- |
| 9. | Consider the university “REVA” is having 100 students where, the details of each student like name, roll no and marks of 3 subjects is to be stored. Design and develop a C program to construct a singly linked list to enter records of different students in list, display the list and calculate the percentage of each student. Also count the number of students passed (scored >40 in all the subjects). |
| 10. | Consider the company “abc” is having 100 employees where, the details of each employee like name and id is to be stored. Design and develop a C program to construct a doubly linked list to enter records of different employees in list, display the list and print the names of all employees with particular employee id. |
| 15. | List of assignments |
| 16. | List of viva questions |

# Program 1: Searching

|  |  |
| --- | --- |
| **1** | **Problem Statement** |
| Consider a Software company “XYZ” with 100 employees. Design and develop a C program to construct an array to store the id of 100 employees and use different searching techniques to search for an employee with particular employee id. | |
| **2** | **Student Learning Outcomes** |
| After successful execution of this program, the student shall able to   * define and create one-dimensional array. * implement different searching algorithms, like linear and binary search. * apply the knowledge of array and searching techniques in problem solving. | |
| **3** | **Design and development of the Program** |
| **3.1** | **Theory** |
| A one-dimensional array (or single dimension array) is a type of linear array. Accessing its elements involves a single subscript which can either represent a row or column index.  Declaration Syntax : **datatype Array\_name[sizeofArray];**  Example: **int array[10];** /\*array can store 10 integer values\*/  **Searching:** Searching technique deals with finding a target in a collection of elements, or determines the target does not exist. Here we consider data in arrays stored in the memory; while in real problems data may be stored in disk files, databases, or even distributed over the Internet. There are basically two type of searching algorithms exists:   * Linear Search * Binary Search   **Linear Search:** A linear search searches an element or value from an array till the desired element or value is not found and it searches in a sequence order. It compares the element with all the other elements given in the list and if the element is matched it returns the position. Linear Search is applied on the unsorted or unordered list when there are fewer elements in a list.  **Binary Search:** Binary Search is applied on the **sorted array or list**. Binary search first compares the value with the elements in the middle position of the array. If the value is matched, then returns the position. If the value is less than the middle element, then it must lie in the lower half of the array and if it's greater than the element then it must lie in the upper half of the array. We repeat this procedure on the lower (or upper) half of the array. Binary Search is useful when there are large numbers of elements in an array. | |

|  |  |
| --- | --- |
| **3.2** | **Algorithm** |
| **Algorithm to search for an element in a Single dimensional array of size “n” using linear search.**  **Linear Search ( Array XYZ, Value id)**  Step 1: Set i to 0 and n to the number of elements in array Step 2: IF i > n then go to step 7  Step 3: IF XYZ[i] = id then go to step 6 Step 4: Set i to i + 1  Step 5: Go to Step 2  Step 6: Print Element id Found at index i and go to step 8 Step 7: Print element not found  Step 8: Exit  **Algorithm to search for an element in a Single dimensional array of size “n” using Binary search .**  **Binary Search(Array XYZ, Value id)**  Step 1: Set n to the number of elements in array, low 0 and highn-1 Step 2: Set mid (low+high)/2  Step 3: While low<=high , Repeat Step 3 to 12 Step 3: IF id=XYZ[mid]  Print Element id Found at index mid Step 4: ELSE IF id<XYZ[mid]  Set highmid-1 Step 5: ELSE  Set lowmid+1 Step 6: Set mid (low+high)/2 Step 7: Exit | |
| **3.3** | **Coding using C Language** |
| **Program to search for the employee with given id in a Single dimensional array of size “n” using liner search.**   1. #include <stdio.h> 2. main()   3. {  4. int XYZ[100], id, i, n, count=0; ***/\*Declaration***\*/ | |

1. printf("Enter the number of employees\t");

6. scanf("%d",&n);

1. printf("Enter %d employee id(s)\t", n);
2. for (i = 0; i < n; i++) ***/\*read the array elements\*/***
3. scanf("%d", &XYZ[i]);
4. printf("Enter the id to search\t");
5. scanf("%d", &id); ***/\*Read the id to be searched\*/***
6. for (i=0;i<n;i++) ***/\*For loop to scan each element of array\*/***

13. {

14. if(XYZ[i]== id) ***/\* IF required element found \*/***

15. {

### /\*Print the location of emp with given searchid\*/

1. printf("employee with %d id is present at location %d.\n", id, i+1);

### count=1; /\*count=1 denotes existance of employee in list\*/

19. }

20. }

1. if (count==0) ***/\*count= =0 denotes absence of employee in list\*/***
2. printf("employee with %d id is not present in array.\t", id);

23. }

**Explanation of the program:**

* Line 1 includes header files using preprocessor directives
* Line 4 declares different variables and array
* Lines 5 and 6 read the number of employees in the array
* Lines 7 to 9 read the employee ids
* Lines 10 and 11 read the id to be searched
* Lines 12 to 20 search for the employee id in the array and prints its location if found
* Lines 21 to 22 print “employee is not present in array” if id does not exist in the array

## Program to search for the employee with given id in a Single dimensional array of size “n” using Binary search.

1. #include<stdio.h>
2. main() 3. {
3. int XYZ[100], id,n,low,high,mid,i; ***/\*Declaration\*/***
4. printf("Enter the number of employees\t"); ***/\*Reading number of employees\*/***

6. scanf("%d",&n);

1. printf("Enter %d employee id(s)\t", n);
2. for (i = 0; i < n; i++) ***/\*Read employee ids\*/***
3. scanf("%d", &XYZ[i]);
4. printf("Enter the id to search\t");
5. scanf("%d", &id); ***/\*Read the id to be searched\*/***
6. low = 0;
7. high = n - 1;
8. mid = (low+high)/2;
9. while (low <= high)

16. {

17. if (XYZ[mid] == id)

18. {

### /\*print the location if employee is found\*/

1. printf("employee with id %d is found at location %d.\n", id, mid+1);
2. break;

22. }

1. else if (id < XYZ[mid])
2. high = mid - 1;
3. else
4. low = mid + 1; 27.

28. mid = (low + high)/2;

29. }

1. if (low > high)
2. printf("employee with id %d is not present in the list.\n", id);

32. }

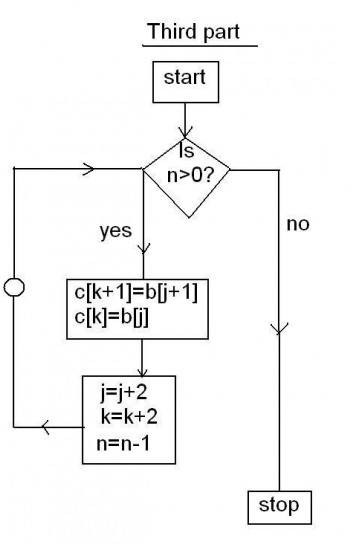
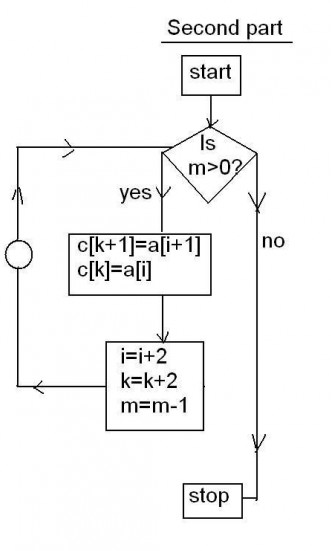
**Explanation of the program:**

* Lines 1 includes header files using preprocessor directives
* Line 4 declares different variables and array
* Lines 5 and 6 read the number of employees in the array
* Lines 7 to 9 read the employee ids
* Lines 10 and 11 read the id to be searched
* Lines 12 to 29 search for the employee id in the array and prints its location if found
* Lines 30 to 31 print “employee is not present in array” if id does not exist in the array

|  |  |
| --- | --- |
| **3.4** | **Expected Results** |
| **Result of Linear Search:**    **Result of Binary Search:** | |

# Program 2: Application of Arrays

|  |  |
| --- | --- |
| **1** | **Problem Statement** |
| Polynomials are used to graph real world curves. For example, roller coaster designers may use polynomials to describe the curves in their rides. Combinations of polynomial functions are sometimes used in economics to do cost analysis. Design and develop a C program to add two  polynomials of any degree using single dimensional array. | |
| **2** | **Student Learning Outcomes** |
| After successful execution of this program, the student shall able to   * Impart the knowledge of array in solving mathematical problems. * Describe different applications of array. * Analyze and create solutions for operations on arrays. | |
| **3** | **Design and development of the Program** |
| **3.1** | **Theory** |
| **Polynomial Addition:** A polynomial **f(x)** is a mathematical expression in the form **anxn + an-1xn- 1 + ... + a0.** The highest exponent of *x* is the **degree** of the polynomial. For example, the degree of **x5 + 3x3 + 4** is **5**.  In a polynomial, an, an-1, ... , a0 are called **coefficients**. If in a polynomial, the coefficients an, an-1,  ... , a1 are all 0, or in other words, the polynomial is in the form of *a0*, we call this polynomial a **constant**. We can add, subtract polynomials by combine the terms in the polynomials with the same powers. For example:  Polynomial addition: (x5 + 3x3 + 4)+(6x6 + 4x3) = 6x6 + x5 + 7x3 + 4 | |
| **3.2** | **Flow Chart** |
| **First part** | |



## Explanation of the Flow Chart:

1. Declare array “a” for storing degree and coefficients of polynomial “first”
2. Declare array “b” for storing degree and coefficients of polynomial “second”
3. Declare array “c” for storing degree and coefficients of polynomial “sum”
4. Declare variable “m” number of terms in first polynomial; “n”number of terms in second polynomial
5. Variables i, j ,k are counters for array a, array b, array c and all initialized to 0;
6. While ( m>0 and n>0) {number of terms in both polynomials is >0}
7. IF(a[i]=b[j]) {degree of first terms in both polynomials are same }
8. c[k+1]=a[i+1]+b[i+1] {add coefficients of first terms of both polynomials}
9. c[k]=a[i] {degree of first term of “c” as degree of first term of “a”}
10. i+=2 {move to next term’s degree in first poly “a”}
11. j+=2 {move to next term’s degree in second poly “b”}
12. m- - {decrease number of terms to be added in “a”}
13. n- - {decrease number of terms to be added in “b”}
14. END OF IF
15. ELSE IF(a[i]>b[j]) {degree of first terms of “a” > degree of first terms of “b”}
16. c[k+1]=a[i+1] {coefficient of first terms of “a” as coefficient of first term of “c”}
17. c[k]=a[i] {degree of first term of “c” will be degree of first term of “a”}
18. i+=2 {move to next term’s degree in first poly “a”}
19. m- - {decrease number of terms to be added in “a”}
20. END OF ELSE IF
21. ELSE(a[i]<b[j]) {degree of first terms of 1st poly < degree of first terms of 2nd poly}
22. c[k+1]=b[j+1] { coefficient of first terms of “b” as coefficient of first term of “c”}
23. c[k]=b[j] {degree of first term of “c” will be degree of first term of “b”}
24. j+=2 {move to next term’s degree in first poly “b”}

|  |  |
| --- | --- |
| 1. n- - {decrease number of terms to be added in “b”} 2. END OF ELSE 3. k+=2 {move to next term of poly “c”} 4. END OF WHILE 5. IF(m>0) {only first polynomial “a” is present} 6. c[k+1]=a[i+1] { coefficient of first terms of “a” as coefficient of first term of “c”} 7. c[k]=a[i] {degree of first term of “c” will be degree of first term of “a”} 8. i+=2 {move to next term’s degree in first poly “a”} 9. k+=2 {move to next term of poly “c”} 10. m - - {decrease number of terms to be added in “a”} 11. END IF 12. IF(n>0) {only second polynomial “b” is present} 13. c[k+1]=b[j+1] { coefficient of first terms of “b” as coefficient of first term of “c”} 14. c[k]=b[j] {degree of first term of “c” will be degree of first term of “b”} 15. j+=2 {move to next term’s degree in first poly “b”} 16. k+=2 {move to next term of poly “c”} 17. n - - {move to next term of poly “c”} 18. END IF | |
| **3.3** | **Coding using C Language** |
| 1. #include<stdio.h> 2. main() 3. {   4. int a[10], b[10], c[10],m,n,k,k1,i,j,x; ***/\*Declaration\*/***  5. printf("\n\tPolynomial Addition\n");  6. printf("\t===================\n");   1. printf("\n\tEnter the no. of terms of the polynomial:"); 2. scanf("%d", &m); ***/\*Read the number of terms in first polynomial “a”\*/*** 3. printf("\n\tEnter the degrees and coefficients:"); 10. for (i=0;i<2\*m;i++) 4. scanf("%d", &a[i]); ***/\*Read the degrees and coefficients of “a”\*/***   ***/\*Print the first polynomial of the above given degrees and co-efficients\*/***   1. printf("\n\tFirst polynomial is:"); 13. k1=0;   14. if(a[k1+1]==1)  15. printf("x^%d", a[k1]);  16. else  17. printf("%dx^%d", a[k1+1],a[k1]);  18. k1+=2; | |

19. while (k1<i)

20. {

21. printf("+%dx^%d", a[k1+1],a[k1]);

22. k1+=2;

23. }

1. printf("\n\n\n\tEnter the no. of terms of 2nd polynomial:");
2. scanf("%d", &n); ***/\*Read the number of terms in second polynomial “b”\*/***
3. printf("\n\tEnter the degrees and co-efficients:"); 27. for(j=0;j<2\*n;j++)

28. scanf("%d", &b[j]); ***/\*Read the degrees and coefficients of “a”\*/***

|  |  |  |
| --- | --- | --- |
| 29. | ***/\*Print the second polynomial of the above given degrees and co-efficients\*/***  printf("\n\tSecond polynomial is:"); | |
| 30. | k1=0; | |
| 31. | if(b[k1+1]==1) | |
| 32. | printf("x^%d", b[k1]); | |
| 33. | else | |
| 34. | printf("%dx^%d",b[k1+1],b[k1]); | |
| 35. | k1+=2; | |
| 36. | while (k1<2\*n) | |
| 37. | { | |
| 38. | printf("+%dx^%d", b[k1+1],b[k1]); | |
| 39. | k1+=2; | |
| 40. | } | |
| 41. | i=0; | |
| 42. | j=0; | |
| 43. | k=0; | |
| ***44.*** | while (m>0 && n>0) ***/\*number of terms in “a” and “b” >0\*/*** | |
| 45. | { | |
| 46. | if (a[i]==b[j]) | ***/\*degree of ith term of “a” and “b” are equal\*/*** |
| 47. | { |  |
| 48. | c[k+1]=a[i+1]+b[j+1]; |  |
| 49. | c[k]=a[i]; |  |
| 50. | m--; |  |
| 51. | n--; |  |
| 52. | i+=2; |  |
| 53. | j+=2; |  |
| 54. | } |  |
| 55. | else if (a[i]>b[j]) |  |
| 56. | { |  |
| 57. | c[k+1]=a[i+1]; |  |

|  |  |  |
| --- | --- | --- |
| 58. |  | c[k]=a[i]; |
| 59. |  | m--; |
| 60. |  | i+=2; |
| 61. | } |  |
| 62. | else |  |
| 63. | { |  |
| 64. |  | c[k+1]=b[j+1]; |
| 65. |  | c[k]=b[j]; |
| 66. | n--; | |
| 67. | j+=2; | |
| 68. | } | |
| 69. | k+=2; | |
| 70. | } | |
| 71. | while (m>0) ***/\*only “a” is present\*/*** | |
| 72. | { | |
| 73. | c[k+1]=a[i+1]; | |
| 74. | c[k]=a[i]; | |
| 75. | k+=2; | |
| 76. | i+=2; | |
| 77. | m--; | |
| 78. | } | |
| 79. | while (n>0) ***/\*only “b” is present\*/*** | |
| 80. | { | |
| 81. | c[k+1]=b[j+1]; | |
| 82. | c[k]=b[j]; | |
| 83. | k+=2; | |
| 84. | j+=2; | |
| 85. | n--; | |
| 86. | } | |

### /\*Print the sum of two polynomials\*/

87. printf("\n\n\n\n\tSum of the two polynomials is:"); 88. k1=0;

89. if (c[k1+1]==1)

90. printf("x^%d", c[k1]);

91. else

92. printf("%dx^%d", c[k1+1],c[k1]);

93. k1+=2;

94. while (k1<k)

95. {

96. if (c[k1+1]==1)

|  |  |
| --- | --- |
| 97. printf("+x^%d", c[k1]);  98. else  99. printf("+%dx^%d", c[k1+1], c[k1]); 100. k1+=2;  101. }  102. }  **Explanation of the program:**   * Lines 8 and 9 read the number of terms in polynomial “first” * Lines 10 to 12 read the degrees and coefficients of polynomial “first” * Lines 14 to 25 print polynomial “first” * Lines 26 and 27 read the number of terms in polynomial “second” * Lines 28 to 31 read the degrees and coefficients of polynomial “second” * Lines 32 to 43 print the polynomial “second” * Lines 44 to 89 calculate the sum of polynomial “first” and polynomial “second” * Line 90 to 104 prints the sum of polynomial “first” and polynomial “second” | |
| **3.4** | **Expected Results** |
|  | |

# Program 3: Structures

|  |  |
| --- | --- |
| **1** | **Problem Statement** |
| Real-life quantities that are naturally described by complex numbers rather than real numbers. Design and develop a C program to add and subtract ‘n’ complex numbers using structure. | |
| **2** | **Student Learning Outcomes** |
| After successful execution of this program, the student shall able to   * Implement structure. * Impart the knowledge of data structures in mathematical problems and solutions. * Create databases using structure. * Implement other data structures like stack, linked list, tree etc using structure. | |
| **3** | **Design and development of the Program** |
| **3.1** | **Theory** |
| **Structure:** Structure is a user defined data type, which allows combining data items of different types. To define a structure, the **struct** statement is used. The struct statement defines a new data type, with more than one member for a program. The format of the struct statement is:  struct [structure tag]  {  member definition; member definition;  ...  member definition;  } [one or more structure variables]; | |
| **3.2** | **Algorithm** |
| **Algorithm to add and subtract two complex numbers using strcuture:** Step 1: Declare structure “complex” with variables “realp and imagp”. Step 2: Declare array of complex numbers ‘comp’  Step 3: Read realp and imagp for complex numbers  Step 4: FOR i0 to i<n Repeat Step 5 where n=Count of complex numbers Step 5: Read realp and imagp for n complex numbers.  Step 6: FOR i0 to i<n Repeat Step 7  Step 7: Print n complex numbers. | |

|  |  |
| --- | --- |
| Step 8: Set sub\_c.realp0, sub\_c.imagp0 Step 9: FOR i0 to i<n Repeat Step 10 and 11  Step 10: sum\_c.realpsum\_c.realp+comp[i].realp  Step 11: sum\_c.imagpsum\_c.imagp+comp[i].imagp  Step 12: Set sub\_c.realpcomp[0].realp , sub\_c.imagpcomp[0].imagp Step 13: FOR i0 to i<n Repeat Step 14 and 15  Step 14: sub\_c.realpsub\_c.realp-comp[i].realp  Step 15: sub\_c.imagpsub\_c.imagp-comp[i].imagp Step 16: Print sum\_c and sub\_c  Step 17: Exit | |
| **3.3** | **Coding using C Language** |
| 1. #include <stdio.h> 2. struct complex ***/\*Declare an structure with tagname ‘complex’\*/***   3. {   1. int realp; ***/\*Member declaration\*/*** 2. int imagp; 3. }comp[50],sum\_c,sub\_c; ***/\*Declare array of complex numbers\*/*** 4. main() 8. { 5. int choice,i,n; 6. printf("Enter the count of complex numbers\n");   11. scanf("%d",&n);  12. printf("Enter %d real parts and imaginary parts:\n",n); 13. for(i=0;i<n;i++)   1. scanf("%d%d",&comp[i].realp,&comp[i].imagp); 2. printf("Enter %d complex numbers are:\n",n); 16. for(i=0;i<n;i++) 3. printf("complex number %d is:%d+%di\n",i,comp[i].realp,comp[i].imagp); 4. while(1)   19. {   1. printf("Choose any one operation\n:"); 2. printf("1: ADD %d COMPLEX NUMBERS\n",n); 3. printf("2: SUBTRACT %d COMPLEX NUMBERS\n",n); 4. scanf("%d",&choice); 5. switch(choice)   25. {   1. case 1: sum\_c.realp=0; 2. sum\_c.imagp=0; | |

|  |  |
| --- | --- |
| 28. for(i=0;i<n;i++)  29. {   1. sum\_c.realp=sum\_c.realp+comp[i].realp; 2. sum\_c.imagp=sum\_c.imagp+comp[i].imagp;   32. }   1. printf("The sum =%d+%di\n",sum\_c.realp,sum\_c.imagp); 2. break; 3. case 2: sub\_c.realp=comp[0].realp; 4. sub\_c.imagp=comp[0].imagp;   37. for(i=0;i<n;i++)  38. {   1. sub\_c.realp=sub\_c.realp-comp[i].realp; 2. sub\_c.imagp=sub\_c.imagp-comp[i].imagp;   41. }   1. printf("The subraction =%d+%di\n",sub\_c.realp,sub\_c.imagp); 2. break; 3. default: printf("Invalid Input\n");   45. }  46. }  47. }  **Explanation of the program**   * Lines 2 to 6 declare a structure and declaration of structure type array and variables. * Lines 10 and 11 read the count of complex numbers * Lines 12 to 14 read the complex numbers * Lines 26 to 34 add complex numbers * Line 35 to 43 subtract complex numbers | |
| **3.4** | **Expected Results** |
|  | |

**Program 4: Stack**

|  |  |
| --- | --- |
| **1** | **Problem Statement** |
| Reversal of string is used to check for palindrome or not. Design and Develop a C program to reverse a string using stack. | |
| **2** | **Student Learning Outcomes** |
| After successful execution of this program, the student shall able to   * study the working of stack * implement the stack operations * apply the knowledge of stack in different applications | |
| **3** | **Design and development of the Program** |
| **3.1** | **Theory** |
| **STACK:** A Stack or LIFO (last in, first out) is an [abstract data type](https://en.wikipedia.org/wiki/Abstract_data_type) that serves as a [collection](https://en.wikipedia.org/wiki/Collection_(computing)) of elements, with two principal operations: **push**, which adds an element to the collection, and **pop**, which removes the last element that was added.    It is a [linear data structure](https://en.wikipedia.org/wiki/Linear_data_structure), or more abstractly a sequential collection in which the push and pop operations occur only at one end of the structure, referred to as the top of the stack. Additionally, a [peek](https://en.wikipedia.org/wiki/Peek_(data_type_operation)) operation may give access to the top. Stack can be implemented either using array or linked list.  **Reversal of string:** | |

|  |  |
| --- | --- |
| **3.2** | **Algorithm** |
| Step 1: Create an empty stack. Step 2: Read String ‘str’  Step 3: FOR i0 to i<Length of string, Repeat Perform Step 4 to 7 to Push characters of string to stack.  Step 4: IF top = STACK\_SIZE-1  Step 5: Print ‘stack overflow’ Step 6: ELSE  Step 7: Set stack[++top]str[i]  Step 8: FOR i0 to i<Length of string, Repeat Step 9 to 13 to Pop all characters from stack and put them back to string.  Step 9: IF top = -1  Step 11: Print stack underflow Step 12: ELSE  Step 13: Set str[i] stack[top--] Step 14: Print string str  Step 15: Exit | |
| **3.3** | **Coding using C Language** |
| 1. #include<stdio.h> 2. #include<string.h> 3. #define MAX 20 4. int top = -1; 5. char stack[MAX]; ***/\*Declaration\*/*** 6. char pop(); ***/\*Function Declaration\*/*** 7. void push(char); ***/\*Function Declaration\*/*** 8. main() 9. { 9. char str[20]; ***/\*array declaration for string\*/*** 10. int i; 11. printf("Enter the string\t"); 12. gets(str); ***/\*Read the string\*/***   ***/\*Push characters of the string str on the stack \*/***   1. for(i=0;i<strlen(str);i++) 2. push(str[i]); | |

16.

17.

***/\*Pop characters from the stack and store in string str \*/***

for(i=0;i<strlen(str);i++) str[i]=pop();

1. printf("\nReversed string is:\t");
2. puts(str);

20. }

***/\*End of main()\*/***

***/\*push function definition\*/***

21. void push(char item)

22. {

23. if(top == (MAX-1))

24. {

1. printf("Stack Overflow\n");
2. return;

27. }

1. stack[++top] =item;
2. } ***/\*End of push()\*/***

***/\*pop function definition\*/***

1. char pop()

31. {

32. if(top == -1)

33. {

1. printf("Stack Underflow\n");
2. return;

36. }

1. return stack[top--];
2. } ***/\*End of pop()\*/***

**Explanation of the program**

* Lines 4 and 5 declare variable
* Lines 6 and 7 declare push and pop functions
* Lines 12 and 13 read the string
* Lines 14 and 15 call push function
* Lines 16 and 17 call pop function
* Lines 18 and 19 print reversed string
* Lines 21 to 29 define function push
* Lines 30 to 38 define function pop

|  |  |
| --- | --- |
| **3.4** | **Expected Result** |
|  | |

# Program 5: Application of Stack

|  |  |
| --- | --- |
| **1** | **Problem Statement** |
| The compilers always convert infix expression into postfix to perform further operations like parsing, lexical analysis etc. Design and Develop a C program to convert an infix expression into postfix using stack. | |
| **2** | **Student Learning Outcomes** |
| After successful execution of this program, the student shall able to   * identify the various application where the stack concepts are used * differentiate between infix/postfix expressions. * implement the expression conversion using stack operations | |
| **3** | **Design and development of the Program** |
| **3.1** | **Theory** |
| **Application of Stack:**   * Parsing * Recursive functions * Expression Evaluation (Postfix evaluation) * Expression conversion   + Infix to prefix   + Infix to postfix   + Postfix to infix   + Prefix to infix * Tower of Hanoi | |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Conversion of infix to postfix expression:** **Expression Representation:**  **Infix Prefix Postfix Infix**  a+b +ab ab+ a+b  a+b\*c +a\*bc abc\*+ a+b\*c (a+b)\*(c-d) \*+ab-cd ab+cd-\* (a+b)\*(c-d)  There is an algorithm to convert an infix expression into a postfix expression. It uses a stack; but in this case, the stack is used to hold operators rather than numbers. The purpose of the stack is to reverse the order of the operators in the expression. It also serves as a storage structure, since no operator can be printed until both of its operands have appeared.  In this algorithm, all operands are printed (or sent to output) when they are read. There are more complicated rules to handle operators and parentheses. Example: A \* B + C becomes A B \* C + The order in which the operators appear is not reversed. When the '+' is read, it has lower precedence than the '\*', so the '\*' must be printed first.  The first column will show the symbol currently being read. The second will show what is on the stack and the third will show the current contents of the postfix string. The stack will be written from left to right with the 'bottom' of the stack to the left. | | | | | | |
|  | S.No. | | current symbol | operator stack | postfix string |  |
|  | 1 | | A |  | A |  |
|  | 2 | | \* | \* | A |  |
|  | 3 | | B | \* | AB |  |
|  | 4 | | + | + | AB\* |  |
|  | 5 | | C | + | AB\*C |  |
|  | 6 | |  |  | AB\*C+ |  |
| **3.2** | | **Algorithm** | | | | |
| **Input:** Infix expression  **Variables used:**s->stack,ch->token read, element->character used to store the element poped  **Output:** Equivalent Postfix Expression  Step1: Stack S Step2: Char ch Step3: Char element  Step4: WHILE(Tokens are Available), Repeat from Step5 to Step12 Step5: ch = Read(Token);  Step6: IF(ch is Operand)  Step7: Print ch ; | | | | | | |

|  |  |
| --- | --- |
| Step 8: End IF Step 9: ELSE  Step10: while(Priority(ch) <= Priority(Top Most Stack)), Repeat Step 10 to 11 Step11: element = Pop(S);  Step12: Print(ele); Step13: Push(S,ch); Step 14: End ELSE  Step15: WHILE(!Empty(S)), Repeat Step 16 and 17 Step16: element = Pop(S);  Step17: Print(ele); Step 18:END WHILE  Step 19: END | |
| **3.3** | **Coding using C Language** |
| 1. #define SIZE 50 ***/\* Size of Stack \*/*** 2. #include<stdio.h> 3. #include<string.h> 4. #include <ctype.h> 5. char s[SIZE]; 6. int top=-1; ***/\* Global declarations \*/*** 7. push(char elem) 8. { ***/\* Function for PUSH operation \*/*** 9. s[++top]=elem;   10. }   1. char pop() 2. { ***/\* Function for POP operation \*/*** 3. return(s[top--]);   14. } 15.  16. int pr(char elem) ***/\* Function for precedence \*/***  17. {  18. switch(elem)  19. {   1. case '#': return 0; 2. case '(': return 1; 3. case '+': 4. case '-': return 2; 5. case '\*': | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 25. |  | case '/': return 3; | | |
| 26. |  | case '$': | | |
| 27. |  | case '^': return 4; | | |
| 28.  29. } | } |  | | |
| 30. main()  31. { | | | | |
| 32. | char infx[50],pofx[50],ch,elem; | | |  |
| 33. | int i=0,k=0; | | |  |
| 34. | printf("\n\nRead the Infix Expression ? "); | | |  |
| 35. | scanf("%s",infx); | | |  |
| 36. | push('#'); | | |  |
| 37. | while( (ch=infx[i++]) != '\0') | | |  |
| 38. | { | | |  |
| 39. | if( ch == '(' ) | | | ***/\* For left parentheses \*/*** |
| 40. | push(ch); | | |  |
| 41. | else if(isalnum(ch)) | | | ***/\* For operand \*/*** |
| 42. | pofx[k++]=ch; | | |  |
| 43.  44. |  | else if( ch == ')')  { | | ***/\* For Right parentheses \*/*** |
| 45. | while( s[top] != '(' ) | | |  |
| 46. | pofx[k++]=pop(); | | |  |
| 47. | elem=pop(); | | | ***/\* Remove ( \*/*** |
| 48. | } | |  | |
| 49. | else | |  | |
| 50. | { | | ***/\* For Operator \*/*** | |
| 51. |  | | while( pr(s[top]) >= pr(ch) ) | |
| 52. |  | | pofx[k++]=pop(); | |
| 53. |  | | push(ch); | |
| 54. | } | |  | |

55. }

1. while( s[top] != '#') ***/\* Pop from stack till empty \*/***
2. pofx[k++]=pop();
3. pofx[k]='\0'; ***/\* Make pofx as valid string \*/***
4. printf("\n\nGiven Infix Expn: %s Postfix Expn: %s\n",infx,pofx);

60. }

## Explanation of the program

* Lines 7 to 29 define the functions used in program

|  |  |
| --- | --- |
| * Lines 34 to 35 read infix expression * Lines 36 to 57 perform conversion algorithm * Line 59 prints the infix and postfix expressions | |
| **3.4** | **Expected Result** |
|  | |

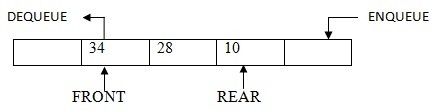
# Program 6: Application of Stack

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **1** | | **Problem Statement** | | | | |
| Evaluation of postfix expressions is done by compilers during the compilation process. Design and Develop a C program to evaluate a postfix expression using stack. | | | | | | |
| **2** | | **Student Learning Outcomes** | | | | |
| After successful execution of this program, the student shall able to   * identify the various application where the stack concepts are used * differentiate between infix/postfix expressions. * implement the postfix expression evaluation using stack | | | | | | |
| **3** | | **Design and development of the Program** | | | | |
| **3.1** | | **Theory** | | | | |
| **Evaluation of postfix Expression:**  Postfix notation (also known as Reverse Polish Notation or RPN in short) is a mathematical notation in which operators follow all of its operands. It is different from infix notation in which operators are placed between its operands. Consider the following example: **234+\*6-**  The exact steps of the algorithm are put in the table below: | | | | | | |
|  | **Input**  **token** | | **Operation** | **Stack contents**  **(TOP=right)** | **Details** |  |
|  | 2 | | Push on the stack | 2 |  |  |
|  | 3 | | Push on the stack | 2, 3 |  |  |
|  | 4 | | Push on the stack | 2, 3, 4 |  |  |
|  | + | | Add | 2, 7 | Pop two values: 3 and 4 and push the result  7 on the stack |  |
|  | \* | | Multiply | 14 | Pop two values: 2 and 7 and push the result  14 on the stack |  |
|  | 6 | | Push on the stack | 14, 6 |  |  |
|  | – | | Subtract | 8 | Pop two values: 14 and 6 and push the  result 8 on the stack |  |
|  | (End of tokens) | | (Return the result) | 8 | Pop the only value 8 and return it |  |

|  |  |
| --- | --- |
| **3.2** | **Algorithm** |
| **Input: A** postfix expression  **Variables used**: op1->operator1, op2->operator2, token-> get next token  **Output :**Result of the postfix expression  Step1: create a new stack  Step2: WHILE(input stream is not empty), Repeat from Step3 to Step10 Step3: token = getNextToken()  Step4: IF(token instanceof operand) Step5: push(token);  Step6: ELSE IF (token instance of operator) Step7: op2 = pop()  Step8: op1 = pop()  Step9: result = calc(token, op1, op2); Step10: push(result)  end ELSE Step11: pop(result) | |
| **3.3** | **Coding using C Language** |
| 1. #define SIZE 50 ***/\* Size of Stack \*/*** 2. #include<stdio.h> 3. #include <ctype.h> 4. int s[SIZE]; 5. int top= -1; ***/\* Global declarations \*/*** 6. void push(int elem) ***/\* Function FOR PUSH operation \*/***   7. {  8. s[++top]=elem; 9. }  ***10.*** int pop() ***/\* Function FOR POP operation \*/***  11. {  12. return(s[top--]);  13. }  14. main()  15. {   1. char pofx[50],ch; 2. int i=0,op1,op2; | |

|  |  |
| --- | --- |
| 1. printf("\n\nRead the Postfix Expression ? "); 2. scanf("%s",pofx);   20. while( (ch=pofx[i++]) != '\0')  21. {   1. if(isdigit(ch)) 2. push(ch-'0'); ***/\* Push the operand \*/*** 3. else 4. { ***/\*pop two operands \*/*** 5. op2=pop(); 6. op1=pop(); 7. switch(ch)   29. {   1. case '+': push(op1+op2); 2. break; 3. case '-': push(op1-op2); 4. break; 5. case '\*': push(op1\*op2); 6. break; 7. case '/': push(op1/op2); 8. break;   38. }  39. }  40. }   1. printf("\n Given Postfix Expn: %s\n",pofx); 2. printf("\n Result after Evaluation: %d\n",s[top]);   43. }  **Explanation of the program**   * Lines 6 to 9 define the function “push” * Lines 10 to 13 define the function “pop” * Lines 18 and 19 read the postfix expression * Lines 20 to 40 evaluate postfix expression * Lines 36-37 print the result | |
|  | **Expected Result** |
|  | |

**Program 7: Queue**



|  |  |
| --- | --- |
| **1** | **Problem Statement** |
| When jobs are entered in system they are stored in the job queue and whenever the processor is free the jobs are moved to ready queue for execution. Consider a college is having a printer. Design and develop a C program to implement job scheduling in the printer.  **Hint:** Store the jobs based on their job id in queue. | |
| **2** | **Student Learning Outcomes** |
| After successful execution of this program, the student shall able to   * implement simple queue * apply various operation performed on queue in problem solving | |
| **3** | **Design and development of the Program** |
| **3.1** | **Theory** |
| **QUEUE:** Queue is a data structure where elements are inserted from one end and deleted from the other end. The end from elements are inserted is called rear end and the end from which elements are deleted is called front end. Since the first element inserted is the first element to be deleted, queue is also called First In First Out data structure.  The various operations that can be implemented in queue are:   1. Enqueue (Insert at rear) 2. Dequeue (Delete from front) 3. Display | |
| **3.2** | **Algorithm** |
| **Input:** The option whether the element is to be inserted , deleted or displayed  **Variables used:** q-> array for queue; front, rear-> variables to show state of queue; MAX-> maximum size of queue; n->option for operation; item->item to be enqueued or dequeued  **Output:** The Output of the operation performed on queue  **Algorithm INSERT\_REAR**  Step 1: IF rear=MAX-1 where MAX=Size of the Queue Step 2: write queue overflow  Step 3: END IF Step 4: ELSE  Step 5: rear=rear+1  Step 6: q[rear]=item where item=item to be inserted Step 7: END ELSE | |

|  |  |
| --- | --- |
| **Algorithm DELETE\_FRONT**  Step 1: IF(front>rear)  Step 2: write queue underflow Step 3: END IF  Step 4: ELSE  Step 5: write q[front] Step 6: front=front+1 Step 7: END ELSE  **Algorithm DISPLAY**  Step 1: IF(front>rear)  Step 2: write queue empty Step 3: END IF  Step 4: ELSE  Step 5: FOR i= rear to front-1 Step 6: write q[i]  Step 7: END FOR  Step 8: END ELSE | |
| **3.3** | **Coding using C Language** |
| 1. #include<stdio.h> 2. #define MAX 10 3. void insert(int item); 4. void delete(); 5. void display(); 6. int jobqueue[MAX],front= -1,rear= -1; 7. main() 8. { 8. int n,job\_id; 9. printf("MENU\n------\n1.ENQUEUE\n2.DEQUEUE\n3.DISPLAY\n4.EXIT\n \n"); 10. while(1)   12. {  13. printf("\nEnter your choice\n");  14. scanf("%d",&n);  15. switch(n)  16. {  ***/\* case to insert a job into queue\*/***   1. case 1: printf("Enter the new job id to be enqueued:"); 2. scanf(“%d”,&job\_id); 3. insert(job\_id); ***/\*Function call to insert\*/*** 4. break;   ***/\*case to delete an element from the queue\*/*** | |

1. case 2: delete();
2. break;

### /\*case to display the jobs waiting in queue\*/

1. case 3: display();
2. break;

### /\*case to exit from loop\*/

1. case 4:
2. break;
3. default:
4. printf("Invalid choice\n");
5. break;

30. }

31. }

32. }

33. void insert(int item)

34. {

35. if(rear<MAX-1) ***/\*Queue is not full\*/***

36. {

37. if(rear==-1) ***/\*Queue is empty\*/***

38. {

1. front=0; ***/\*increase front by 1\*/***
2. rear=0; ***/\*increase rear by 1\*/***
3. jobqueue[rear]=item; ***/\*insert job at rear\*/***

42. }

1. else
2. jobqueue[++rear]=item; ***/\*increase rear by 1 and insert item\*/***

45. }

1. else
2. printf("Job Queue Overflow\n"); ***/\*Queue full\*/***

### }/\*End of insert()\*/

1. void delete()

50. {

51. if(front>=0) ***/\*Queue is not empty\*/***

52. {

### /\*dequeue element from front\*/

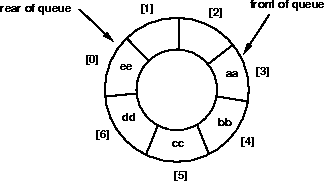
53. printf("The job with id %d is to be served first from queue",jobqueue[front]);

|  |  |  |
| --- | --- | --- |
| 54. | if(front==rear) | ***/\*only one element in queue\*/*** |
| 55. | { |  |
| 56. | front= -1; | ***/\*no element left to be deleted\*/*** |
| 57. | rear= -1; | ***/\*no element in queue\*/*** |
| 58. | } |  |
| 59. | else |  |
| 60. | front++; | ***/\*increase front by 1\*/*** |
| 61. | } |  |
| 62. | else |  |

63. printf("Job Queue Underflow! There are no jobs in queue\n"); ***/\*Queue empty\*/***

|  |  |
| --- | --- |
| 1. }***/\*End of delete()\*/*** 2. void display()   66. {   1. if((front= = -1)&&(rear= = -1)) ***/\*Queue empty\*/*** 2. printf("The queue is empty\n"); 3. else   70. {   1. int i; 2. printf("The jobs queued in the job queue are :"); 3. for(i=front;i<=rear;i++) ***/\*traverse elements from front to rear\*/*** 4. printf("%d\t",jobqueue[i]); ***/\*print elements from queue\*/***   75. }  76. }***/\*End of display()\*/***  **Explanation of the program**   * Lines 3 to 5 declare functions * Line 6 declares the queue * Lines 17 to 20 enqueue job with particular id by calling push function * Lines 21 to 22 delete( process) job with particular id by calling pop function * Lines 23 to 24 display the job queue * Lines 33 to 48 define the function “insert” * Lines 49-64 define the function “delete” * Lines 65-76 define the function “display” | |
|  | **Expected Result** |
|  | |

# Program 8: Circular Queue



|  |  |
| --- | --- |
| **1** | **Problem Statement** |
| In a machine gun bullets are always inserted in circular queue. Design and develop a C program to process bullets in machine gun.  **Hint:** Bullet insertion and firing (deletion) can be implemented using circular queue. Bullets are given with specific id. | |
| **2** | **Student Learning Outcomes** |
| After successful execution of this program, the student shall able to   * Define circular queue * Implement the operations performed on circular queue and apply the same for problem solving. | |
| **3** | **Design and development of the Program** |
| **3.1** | **Theory** |
| In case of ordinary queue, rear insertion is not possible even if space is available at the front end. This is because before inserting an element in ordinary queue, it is tested whether rear is greater than the maximum size of the queue. If so, a message is printed stating queue size full and cannot be inserted. This disadvantage can be overcome using circular queue. **Circular queue** is a linear data structure. It follows First In First Out principle. In circular queue the last node is connected back to the first node to make a circle. Elements are added at the rear end and the elements are deleted at front end of the queue. Both the front and the rear pointers points to the beginning of the array. It is also called as “Ring buffer”. | |

|  |  |
| --- | --- |
| **3.2** | **Algorithm** |
| **Input:** Option whether the element is to be inserted, deleted or displayed  **Variables used:** cq->circular queue; front,rear->basic variables to show state of circular queue; MAXSIZE-> maximum size of Circular queue; num, item->item to be inserted or deleted  **Output:** Implementation of circular queue  **Algorithm: Insert**  Step 1: IF(front=(rear+1)%QUEUE\_SIZE) Step 2: write queue overflow  Step 3: END IF Step 4: ELSE  Step 5: IF(front== -1)  Step 6: front=rear=0; Step 7: END IF  Step 8: END ELSE Step 9: ELSE  Step 10: rear=(rear+1)%MAXSIZE; where MAXSIZE=Size of the queue Step 11: cq[rear]=item;  Step 12: END ELSE  **Algorithm: Delete**  Step 1: IF(front == -1)  Step 2: write CIRCULAR QUEUE IS UNDERFLOW Step 3: END IF  Step 4: ELSE  Step 5: item=cq[front];  Step 6: cq[front]=0; Step 7: END ELSE  Step 8: IF(front==rear)  Step 9: front=rear=-1; Step 10:END IF  Step 11:ELSE  Step 12: front = (front+1)%MAXSIZE;  Step 13: write DELETED ELEMENT FROM QUEUE IS :a Step 14:END ELSE | |

|  |  |
| --- | --- |
| **3.3** | **Coding using C Language** |
| 1. #include <stdio.h> 2. #include<stdlib.h> 3. #define MAXSIZE 5 4. int cq[MAXSIZE]; /\*initialize circular queue\*/ 5. int front=-1,rear=-1; 6. void add(int); 7. void del(); 8. void display(); 9. int main()   10. { 11.   1. int ch,i,num; 2. printf("\nCircular Queue implementation for machine gun execution"); 3. printf("\n\nMAIN MENU\n 1.INSERTION\n 2.DELETION\n 3.DISPLAY\n 4.EXIT"); 4. while(1)   16. {   1. printf("\n\nENTER YOUR CHOICE OF OPERATION: "); 2. scanf("%d",&ch); 3. switch(ch)   20. {   1. case 1: 2. printf("\n\nENTER THE QUEUE ELEMENT(BULLET ID) TO INSERT: "); 3. scanf("%d",&num); /\*read element to be inserted\*/ 4. add(num); /\*call function add\*/ 5. break; 6. case 2: 7. del(); /\*call function del\*/ 8. break; 9. case 3: 10. display(); /\*call function display\*/ 11. break; 12. case 4: 13. exit(0); 14. default: printf("\n\nInvalid Choice . ");   35. }  36. }  37. }   1. /\*function to add an element into queue\*/ 2. void add(int item)   40. {  41. if(front ==(rear+1)%MAXSIZE) /\*Queue is full\*/  42. {  43. printf("\n\nCIRCULAR QUEUE(GUN)IS FULL"); | |

|  |  |  |  |
| --- | --- | --- | --- |
| 44. | } |  | |
| 45. | else |
| 46. | { |
| 47. |  | if(front==-1) /\*Queue empty\*/ | |
| 1. front=rear=0; /\*increase front and rear by 1\*/ 2. else | | | |
| 50. | rear=(rear+1)%MAXSIZE; | | /\*move rear one place to the right\*/ |
| 51. | cq[rear]=item; | | /\*insert element at rear\*/ |

52. printf("\n\nRear = %d Front = %d ",rear,front); /\*print front and rear value \*/

53. }

54. }

1. /\*function to delete an element from the queue\*/
2. void del()

57. {

1. int a;
2. if(front == -1) /\*Queue empty\*/
3. printf("\n\nCIRCULAR QUEUE(GUN) IS EMPLTY");
4. else

62. {

1. a=cq[front]; /\*dequeue element from front and assign it to a\*/
2. cq[front]=0; /\*assign 0 at front\*/
3. if(front==rear) /\*only one element in queue\*/
4. front=rear=-1; /\*move front one place to its left\*/
5. else
6. front = (front+1)%MAXSIZE; /\*move front one place to its right\*/
7. printf("\n\nfIRED BULLET ID FROM QUEUE(GUN) IS : %d ",a); /\*print deleted element\*/
8. printf("\n\nRear = %d Front = %d ",rear,front); /\*print the values of front and rear\*/

71. }

72. }

73. void display() /\* function to display the queue elements\*/

74. {

|  |  |  |
| --- | --- | --- |
| 75. | int i; |  |
| 76. | if(front<rear) |
| 77. | { |
| 78. | for(i=front;i<=rear;i++) | /\*traverse queue elements from front to rear\*/ |
| 79. | printf("%d\t",cq[i]); | /\*print elements of queue\*/ |
| 80. | } |  |
| 81. | else |  |
| 82. | { |  |
| 83. | for(i=0;i<MAXSIZE;i++) |  |
| 84. | printf("%d\t",cq[i]); |  |
| 85.  86. } | } /\*elements of queue\*/ | |

|  |  |
| --- | --- |
| **Explanation of the program**   * Line 4 and 5 declare circular queue and other variables. * Lines 6 to 8 declare the prototype of insert, delete and display functions. * Lines 24 to 31 call for insert, delete and display functions. * Lines 39 to 54 define the function “Insert” to insert elements into the queue * Lines 56 to 72 define the function “del” to delete elements from the queue * Lines 74 to 91 define the function “”display” to display the elements of the queue | |
| **3.4** | **Expected Results** |
|  | |

# Program 9: Singly Linked List

|  |  |
| --- | --- |
| **1** | **Problem Statement** |
| Consider the University “REVA” is having 100 students where, the details of each student like name, roll no and marks of 3 subjects is to be stored. Design and develop a C program to construct a singly linked list to enter records of different students in list, display the list and calculate the percentage of each student. Also count the number of students passed (scored >40 in all the subjects). | |
| **2** | **Student Learning Outcomes** |
| After successful execution of this program, the student shall able to   * implement single linked list. * differentiate between array and linked list. * implement various operations in single linked list. * create and manage databases using linked list. | |
| **3** | **Design and development of the Program** |
| **3.1** | **Theory** |
| A linked list is a [data structure](https://en.wikipedia.org/wiki/Data_structure) consisting of a group of [nodes](https://en.wikipedia.org/wiki/Node_(computer_science)) which together represent a sequence. In singly linked list each node is composed of data and a pointer (link) to the next node in the sequence. This structure allows for efficient insertion or removal of elements from any position in the sequence.  **Operations on Linked list:**   * Creation of a linked list * Traversal * Searching an element * Insertion of an element at different positions. * Deletion of an element from different positions. * Reversing a linked list | |
| **3.2** | **Algorithm** |
| **Input :** choice to insert , delete or display the values of nodes  **Variables used:**t->node,start->pointer which points to the first node,n->data  **Output:** Singly linked list implemented | |

|  |  |
| --- | --- |
| **INSERTION AT BEGINNING**  Step 1: tempnew node  Step 2: enter info to be inserted Step 3: read n  Step 4:temp->infon Step 5: temp->link start Step 6: start temp  **INSERTION AT END**  Step 1: pstart  Step 2:Repeat step 3 untill p->next=NULL Step 3: pp->next  Step 4: t->nextNULL  Step 5: p->nextt  **DELETION AT POSITION**  Step 1: IF pos=1 then Step 2: tempstart  Step 3: starttemp->link  Step 4: ELSE  Step 5: pstart  Step 6: FOR i0 to i < pos-2 Repeat Steps 8 Step 7: pp->link  END FOR  Step 8: tempp->link  Step 9: p->linktemp->link  **DISPLAY**  Step 1: pstart  Step 2: WHILE(p!=NULL)  Step 3: print p->info  Step 4: pp->link | |
| **3.3** | **Coding using C Language** |
| 1. #include<stdio.h> 2. #include<stdlib.h> 3. struct REVA ***/\*creation of linked list\*/***   4. { | |

1. char name[20];
2. int rollno;
3. float sub1,sub2,sub3;
4. struct REVA \*link;
5. }\*start=NULL;

### /\*Function definition\*/

1. struct REVA \*addatbeg(struct REVA \*start);
2. struct REVA \*addatend(struct REVA \*start);
3. struct REVA \*delatpos(struct REVA \*start,int pos);
4. void display(struct REVA \*start);
5. void percentandcount(struct REVA \*start);
6. main()

16. {

1. int choice,pos;
2. printf("1Add Student Info in the beginning\n");
3. printf("2 Add Student Info in the last\n");
4. printf("3Delete Student Info from any position\n");
5. printf("4Display Students Info in the list\n");
6. printf("5Print percentage of each student and pass count\n");
7. printf("6Exit\n");
8. while(1)

25. {

1. printf("\nChoose an operation:\n");
2. scanf("%d",&choice);
3. switch(choice)

29. {

1. case 1: start=addatbeg(start);
2. break;
3. case 2: start=addatend(start);
4. break;
5. case 3: printf("Enter the position\t:");
6. scanf("%d",&pos);
7. start=delatpos(start,pos);
8. break;
9. case 4: display(start);
10. break;
11. case 5: percentandcount(start);
12. break;
13. case 6: exit(0);

43. }

44. }

### }/\*End of main()\*/

***/\*Function to insert at beginning\*/***

1. struct REVA \*addatbeg(struct REVA \*start)

47. {

1. struct REVA \*temp;
2. temp=(struct REVA \*)malloc(sizeof(struct REVA));
3. printf("Enter student name , roll no and marks for 3 subjects\t:");
4. scanf("%s%d%f%f%f",temp->name,&temp->rollno,&temp->sub1,&temp-

>sub2,&temp->sub3);

1. temp->link=start;
2. start=temp;
3. return start;

### } /\*End of function\*/

***/\*Function to insert at last\*/***

1. struct REVA \*addatend(struct REVA \*start)

57. {

1. struct REVA \*temp;
2. temp=(struct REVA \*)malloc(sizeof(struct REVA));
3. printf("Enter student name , roll no and marks for 3 subjects\t:");
4. scanf("%s%d%f%f%f",temp->name,&temp->rollno,&temp->sub1,&temp->sub2, &temp->sub3);
5. struct REVA \*p=start;
6. while(p->link!=NULL)

64. {

65. p=p->link;

66. }

1. temp->link=NULL;
2. p->link=temp;
3. return start;

### }/\* End of function\*/

***/\*Function to delete from a position\*/***

1. struct REVA \*delatpos(struct REVA \*start,int pos)

72. {

1. int i;
2. struct REVA \*temp;
3. if(pos==1)

76. {

1. temp=start;
2. start=temp->link;

79. }

80. else

81. {

82. struct REVA \*p=start;

83. for(i=0;i<pos-2;i++)

84. {

85. p=p->link;

86. }

1. temp=p->link;
2. p->link=temp->link;

89. }

1. free(temp);
2. return start;

### }/\* End of function\*/

***/\*Function to display the list\*/***

1. void display(struct REVA \*start)

94. {

1. struct REVA \*p=start;
2. while(p!=NULL)

97. {

1. printf("%s,%d,%f,%f,%f-->",p->name,p->rollno,p->sub1,p->sub2,p->sub3);
2. p=p->link;

100. }

101. }***/\* End of function\*/***

***/\*Function to display percentage and pass count\*/***

|  |  |
| --- | --- |
| 102. | void percentandcount(struct REVA \*start) |
| 103. | { |
| 104. | struct REVA \*p=start; |
| 105. | float per; |
| 106. | int count=0; |
| 107. | while(p!=NULL) |
| 108. | { |
| 109. | per=(p->sub1+p->sub2+p->sub3)/3; |
| 110. | printf("%f-->",per); |
| 111. | if(p->sub1>40&&p->sub2>40&&p->sub3>40) |
| 112. | count++; |
| 113. | p=p->link; |
| 114. | } |
| 115. | printf("Total Number of students passed:%d\n",count); |
| 116. | }***/\*End of function\*/*** |

|  |  |
| --- | --- |
| **Explanation of the program**   * Lines 3 to 9 declare structure for node * Lines 10 to 14 declare all functions * Lines 30 to 40 call functions * Lines 46 to 55 define the function “addatbeg” * Lines 56 to 70 define the function “addatend” * Lines 71 to 92 define the function “delatpos” * Lines 93 to 101 define function “display” * Lines 102 to 116 define the function “percentandcount” | |
| **3.4** | **Expected Results** |
|  | |

# Program 10: Doubly Linked List

|  |  |
| --- | --- |
| **1** | **Problem Statement** |
| Consider the company “abc” is having 100 employees where, the details of each employee like name and id is to be stored. Design and develop a C program to construct a doubly linked list to enter records of different employees in list, display the list and print the names of all employees with particular employee id. | |
| **2** | **Student Learning Outcomes** |
| After successful execution of this program, the student shall able to   * implement doubly linked list. * differentiate between array and linked list. * implement various operations in doubly linked list. * create and manage databases using linked list. | |
| **3** | **Design and development of the Program** |
| **3.1** | **Theory** |
| In a doubly linked list, each node contains, besides the next-node link, a second link field pointing to the 'previous' node in the sequence. The two links may be called ‘forward’ and 'backward', or 'next' and 'prev'(previous).  **Operations on Linked list:**   * Creation of a linked list * Traversal * Searching an element * Insertion/Deletion of an element at different positions. * Reversing a linked list | |
| **3.2** | **Algorithm** |
| **Input :** choice to insert , delete or display the values of nodes  **Variables used:**temp->node,start->pointer which points to the first node,n->data  **Output:** Doubly linked list implemented  **ADDING A NEW VALUE AS A NEW NODE TEMP:**  Step 1: tempnew node  Step 2: enter info to be inserted Step 3: read n  Step 4: temp->infon Step 5:temp->nextNULL  Step 6: temp->prevNULL | |

|  |  |
| --- | --- |
| **INSERTION AT BEGIN**  Step 1: temp->prev NULL Step 2: temp->nextstart  Step 3: starttemp  **INSERTION AT END**  Step 1: pstart  Step 2:Repeat step 3 untill p->next=NULL Step 3: pp->next  Step 4: temp->nextNULL  Step 5: temp->prevp  Step 6: p->nexttemp  **DISPLAY**  Step 1: pstart  Step 2: WHILE p=NULL Repeat Step 3 and 4 Step 3: print p->info  Step 4: pp->next  **SEARCH**  Step 1: pstart  Step 2: pos0  Step 3: WHILE p=NULL Repeat Step 3 and 4  Step 4: pospos+1  Step 5: IF p->info = search key Step 6: print “pos”  Step 7: pp->next | |
| **3.3** | **Coding using C Language** |
| 1. #include<stdio.h> 2. #include<stdlib.h>   ***/\*structure definition\*/***   1. struct abc 4. { 2. char name[20]; 3. int empid; 4. struct abc \*prev; 5. struct abc \*next; 6. }\*start=NULL;   ***/\*Function Declaration\*/***   1. struct abc \*addatbeg(struct abc \*start); 2. struct abc \*addatend(struct abc \*start); | |

1. void display(struct abc \*start);
2. void search(struct abc \*start,int key);
3. main()

15. {

1. int choice,pos,key;
2. printf("1->Add Employee Info in the beginning\n");
3. printf("2->Add Employee Info in the last\n");
4. printf("3->Display Etudents Info in the list\n");
5. printf("4->Search for an Employee\n");
6. printf("5->Exit\n");
7. while(1)

23. {

1. printf("\nChoose an operation:\n");
2. scanf("%d",&choice);
3. switch(choice)

27. {

1. case 1: start=addatbeg(start);
2. break;
3. case 2: start=addatend(start);
4. break;
5. case 3: display(start);
6. break;
7. case 4: printf("Enter the employee id to be searched\t:");
8. scanf("%d",&key);
9. search(start,key);
10. break;
11. case 5: exit(0);

39. }

40.

41. }

1. getch();

### }/\*End of main()\*/

***/\* Function to add element in beginning \*/***

1. struct abc \*addatbeg(struct abc \*start)

45. {

1. struct abc \*temp;
2. temp=(struct abc \*)malloc(sizeof(struct abc));
3. printf("Enter employee name and id\t:");
4. scanf("%s%d",temp->name,&temp->empid);
5. temp->prev=NULL;
6. temp->next=start;
7. start=temp;
8. return start;

### }/\*End of addatbeg()\*/

***/\* Function to add element at end \*/***

1. struct abc \*addatend(struct abc \*start)

56. {

1. struct abc \*temp;
2. temp=(struct abc \*)malloc(sizeof(struct abc)); ***/\*node creation\*/***
3. printf("Enter employee name and id\t:");
4. scanf("%s%d",temp->name,&temp->empid);
5. struct abc \*p=start;
6. while(p->next!=NULL)

63. {

64. p=p->next;

65. }

1. temp->next=NULL;
2. temp->prev=p;
3. p->next=temp;
4. return start;

### }/\*End of addatend()\*/

***/\* Function to display elements (employees information) \*/***

1. void display(struct abc \*start)

72. {

1. struct abc \*p=start;
2. while(p!=NULL)

75. {

1. printf("%s,%d-->",p->name,p->empid);
2. p=p->next;

78. }

### }/\*End of display()\*/

***/\* Function to search an employee with particular id \*/***

1. void search(struct abc \*start,int key)

81. {

1. struct abc \*p=start;
2. int pos=0;
3. while(p!=NULL)

85. {

1. pos++;
2. if(p->empid==key)

88. {

|  |  |
| --- | --- |
| 1. printf("employeeid=%d, name=%s, position=%d\n",p->empid,p->name,pos); 2. break;   91. }   1. else 2. p=p->next;   94. }   1. if(pos==0) 2. printf("employee with id %d is not present in the list\n",p->empid); 3. }***/\*End of search()\*/***   **Explanation of the program**   * Lines 3 to 9 declare structure for node * Lines 10 to 14 declaration of all functions * Lines 30 to 38 call functions * Lines 39 to 40 read search key * Lines 50 to 60 define the function “addatbeg” * Lines 61 to 76 define the function “addatend” * Lines 101 to 109 define function “display” * Lines 110 to 127 define the function “search” | |
| **3.4** | **Expected Results** |
|  | |

# List of Lab Assignments

* Design and develop a C program to create an array to store the information of employees like id, name and salary and use different searching techniques to return the id and name of all employees with given salary using Linear Search.
* Design and develop a C program to print the Pascal Triangle as given below using array. 1

1 1

1 2 1

1 3 3 1

* Design and develop a C program to store and retrieve customer database in a bank. Note: Customer database can include Customer id, Name, Account Number etc.
* Design and Develop a C program to convert a decimal number to binary using a stack.
* Design and develop a c program to convert an infix expression into prefix.
* Design a C program to print all the prime factors of a number in descending order using stack. For example the prime divisors of 450 are 5,5,3,3,2
* Design and implement a program to create a stack using queue.
* Construct a circular queue to create a employee database like name, id and perform all the operations – insert, delete, display.
* Construct a single linked list to store the employee database like employee name, id and salary and search for an employee with a particular salary.
* Construct a doubly linked list to store student database and print all the elements in reverse order.
* Design and develop a C program to implement Ackermann’s function A(m, n) using recursive technique:

1) If m=0, A(m, n)=n+1

2) If m>0 and n=0, A(m, n)=A(m-1, 1)

3) If m>0 and n>0, A(m, n)=A(m-1, A(m, n-1))

* Design and develop a C program to find the minimum and maximum element in the binary search tree
* Construct a C program to search for a particular element in graph.
* Implement the above program using selection sort.

# List of Viva Questions

* What are the I/O functions required to read and print the array elements?
* What will happen if you assign a value to an array element whose subscript exceeds the size of array?
* If you pass an array as an argument to a function, what actually gets passed?
* Which searching technique is more efficient when given array is sorted and why?
* Brief the logic of linear search.
* Brief the logic of binary search.
* What are the application of arrays?
* What are the disadvantage of arrays?
* What is meant by a “polynomial” and “degree of polynomial” ?
* Can array be passed as an argument to a function?
* What are the similarities and differences between structures and union?
* Can a structure be defined within another structure?
* Write a function to initialize a structure variable?
* What are the applications of stack ?
* Write functions for push, pop and peek operation for stack?
* Explain Tower of Hanoi and how is it related to STACK?
* Convert the expression ((A + B) \* C - (D - E) ^ (F + G)) to equivalent Prefix notation.
* Convert the expression ((A + B) \* C - (D - E) ^ (F + G)) to equivalent postfix notation.
* Convert the postfix expression 2 3 8 \* + 4 48 4 2 + / 6 \* + - into prefix expression.
* Convert the postfix expression 2 3 8 \* + 4 48 4 2 + / 6 \* + - into infix expression
* Evaluate 2 3 8 \* + 4 48 4 2 + / 6 \* + - postfix expression.
* Explain the algorithm to evaluate postfix expression.
* What are the applications of queue?
* What are the values of FRONT and REAR in different cases of Queue?
* Explain the logic behind insert and delete function.
* What are the applications of a queue and circular queue?
* What is a circular queue and doubly ended queue?
* Explain basic conditions for enqueue and dequeue operations in circular queue?
* What are the differences between array and linked list implementation of a circular queue?
* Is a Linked List linear or Non-linear data structure?
* What are the advantages and disadvantages of a linked list?
* Write a function to count the number of occurrences of an element in single linked list
* How do you insert a record between two nodes in double-linked list?
* Is a Linked List linear or Non-linear data structure?
* What are the advantages and disadvantages of a linked list?
* Write a function to count the number of occurrences of an element in single linked list
* How do you insert a record between two nodes in double-linked list?
* Give some real life example where linked list is used.
* What is the data structure used to implement the recursive technique?
* What do you mean by recursive definition?
* What are the advantages of recursion over iteration?
* Name some applications of recursion?
* List out the few applications of binary tree?
* Which of the data structures Array, Linked list, Stack and Queue is efficient in constructing a tree?
* Construct a binary tree from inorder 35,26,93,21,68 and preorder 68,21,93,26,35.
* Write a function to search a key in a Binary Search Tree.
* Write preorder, postorder, inorder and level order traversal of a given tree.
* What is sorting?
* Explain the logic of bubble sort?
* What is the time complexity for bubble sort?
* How many sorting algorithms are present?