**SSN College of Engineering, Kalavakkam**

**Department of Computer Science and Engineering**

**III Semester - CSE**

# UCS 1312 Data Structures Lab Laboratory

|  |  |
| --- | --- |
| **Academic Year: 2019-2020** | **Batch: 2018-2022** |

**Exercise 1: Student Information System using Array Implementation of list ADT**

Create a StudentADT with the following fields

Set of students where each student contains Regno. Name, Mark1, Mark2,Mark3,

Total, Result

StudentList ADT has the implementations for the following operations

1. Insert a record in the front of the list

void insertFront(listADT l,student s)

1. Insert a record at the end of the list

void insertEnd(listADT l,student s)

1. Insert a record after a given Regno in the list

void insertRegNo(listADT l, student s, int regNum)

1. Insert a record after a given Regno in the list

student searchRegNo(listADT l, int regNum)

1. List the records of students based on his register number

listADT searchName(listADT l, char name[])

1. Delete a given student record given his register number and display all the details

void delete(listADT l, int regNum)

1. Calculate the Total and update the Result field

void computeResult(listADT l)

1. List the students who have passed

student\* listResult(listADT l)

1. List the students how many have secured FirstClass

int listClass(listADT l)

In order ro implement this Student Information System,

* It is necessary to create a file that has StudentList ADT and implementation of above-mentioned functions
* Another file will be created with only function prototypes
* One more file will be created to write the Student Information System using the StudentList ADT

**Note: Submit the code along with the output within the deadline**

**SSN College of Engineering, Kalavakkam**

**Department of Computer Science and Engineering**

**III Semester - CSE**

# UCS 1312 Data Structures Lab Laboratory

|  |  |
| --- | --- |
| **Academic Year: 2019-2020** | **Batch: 2018-2022** |

**Exercise 1: Polynomial manipulation using Linked List**

Create a PolynomialADT with the following fields

Exponent, Coefficient and a pointer to the next node

Polynomial ADT has the implementations for the following operations to

1. Input a polynomial through insertion at the front

void insertFront(polyADT p,term t)

1. Input a polynomial through insertion at the end

void insertEnd(polyADT p,term t)

1. Input a polynomial after a term

void insertAfterTerm(polyADT p, term t, int exp)

1. Add two polynomials

polyADT polyAdd(polyADT p1, polyADT p2)

1. Multiply two polynomials

polyADT polyAdd(polyADT p1, polyADT p2)

1. Find the degree of polynomial

void polyDegree(polyADT p)

1. Evaluate a polynomial

int polyEvaluate(polyADT p)

1. Simplifying the polynomial – Combining like terms

polyADT polySimplify(polyADT p)

In order ro implement this Student Information System,

* It is necessary to create a file that has polyADT and implementation of above-mentioned functions
* Another file will be created with only function prototypes
* One more file will be created to write the Student Information System using the polyADT

**Note: Submit the code along with the output within the deadline**

**SSN College of Engineering, Kalavakkam**

**Department of Computer Science and Engineering**

**III Semester - CSE**

# UCS 1312 Data Structures Lab Laboratory

|  |  |
| --- | --- |
| **Academic Year: 2019-2020** | **Batch: 2018-2022** |

**Exercise 3: Student Information System using Doubly Linked List**

Create a personADT with the following fields

Person Identification Number (PIN) [should be a unique random number generated for each person while inserting the person’s record], Name, Age, Sex, Street Address, City, Mobile,

personList ADT has the implementations for the following operations

1. Insert a record in the front of the list

void insertFront(listADT l, person p)

1. Insert a record at the end of the list

void insertEnd(listADT l, person p)

1. Insert a record after a given PIN in the list

void insertPIN(listADT l, person p, int pin)

1. List the senior persons based on their age

listADT seniorPerson(listADT l)

1. List the adults based on their age

listADT adultPerson(listADT l)

1. List the persons based on the given location

listADT locatePerson(listADT l, char \* s)

1. Sort the persons based on their age in both ascending and descending order based on their age and display

void listSort(listADT l)

1. Number of persons in the list

int noPersons(listADT l)

1. Give the ratio of male versus female, where ratio is the structure containing male and female. (For example, it should be printed as 4:3)

Ratio adultPerson(listADT l)

1. Display the persons in the list

void listPersons(listADT l)

In order ro implement this Student Information System,

* It is necessary to create a file that has PersonList ADT and implementation of above-mentioned functions
* Another file will be created with only function prototypes
* One more file will be created to write the application using the PersonList ADT

**Note: Submit the code along with the output within the deadline**

**SSN College of Engineering, Kalavakkam**

**Department of Computer Science and Engineering**

**III Semester - CSE**

# UCS 1312 Data Structures Lab Laboratory

|  |  |
| --- | --- |
| **Academic Year: 2019-2020** | **Batch: 2018-2022** |

**Exercise 4: Evaluation of expression using its postfix form and stack**

The structure stack consists of character array and top. Implement stack using array with the following methods.

* void push(Stack \*S, char x) – Push an element into the stack
* char pop(Stack \*S) – Pop an element from the stack
* void disp(Stack \*S) – Display elements from the stack
* int isEmpty(Stack \*S) – Check whether the stack is empty
* int isFull(Stack \*S) – Check whether the stack is full

Note:

1. Implement stack with the specified operations in stack.h
2. Check the stack by writing application program in c
3. Write a program in with the following functions

* To convert the given infix expression (without involving parenthesis) into its postfix form using the stack
  + char \* infixtoPostfix(char \*str, struct stack \*s)
* To evaluate the expression considering its postfix form
  + int evaluateExp(char \*postfix, struct stack \*s)

**SSN College of Engineering, Kalavakkam**

**Department of Computer Science and Engineering**

**III Semester - CSE**

# UCS 1312 Data Structures Lab Laboratory

|  |  |
| --- | --- |
| **Academic Year: 2019-2020** | **Batch: 2018-2022** |

**Exercise 5: Implementation of Linear, Circular Queue and Application of Circular Queue**

The structure Queue consists of integer array, front and rear. Implement both linear and Circular Queue using array with the following methods.

* void enqueu(Queue \*Q, int x) – Insert an element into the queue
* int dequeue(Queue \*Q) – Dequeue an element from the queue
* void disp(Queue \*Q) – Display elements from the Queue
* int isEmpty(Queue \*Q) – Check whether the queue is empty
* int isFull(Queue \*Q) – Check whether the queue is full

Note:

1. Implement linear queue with the specified operations in linearqueue.h
2. Implement circular queue with the specified operations in circularqueue.h
3. Check the linear queue by writing application program in lqueueapp.c
4. Check the circular queue by writing application program in cqueueapp. C

Application of Circular Queue

1. Modify the circular queue to contain job number and the cpu burst time
2. Instantiate 2 circular queues Q1 and Q2
3. Insert circular queue with the following contents

(J1,2), (J2,4), (J3,8), (J4,5), (J5,2), (J6,7), (J7,4), (J8,3) (J9,6) & (J10,6)

1. Insert the job into the circular queue whichever is empty. If it is not empty, insert the job into the queue whichever is having minimum average time
2. Display the jobs waiting in both the queues along with their cpu burst time.

**SSN College of Engineering, Kalavakkam**

**Department of Computer Science and Engineering**

**III Semester - CSE**

# UCS 1312 Data Structures Lab Laboratory

|  |  |
| --- | --- |
| **Academic Year: 2019-2020** | **Batch: 2018-2022** |

**Exercise 6: Construction and Evaluation of Expression Tree**

1. Get an expression and convert that into postfix notation using the conversion algorithm
2. Using stack, construct expression tree from the postfix notation

while(not the end of the expression)

{

if(the next symbol in the expression is an operand)

{

create a node for the operand ;

push the reference to the created node onto the stack ;

}

if(the next symbol in the expression is a binary operator)

{

create a node for the operator ;

pop from the stack a reference to an operand ;

make the operand the right subtree of the operator node ;

pop from the stack a reference to an operand ;

make the operand the left subtree of the operator node ;

push the reference to the operator node onto the stack ;

}

}

1. Evaluate the expression from evaluation tree using the algorithm

Evaluate(ExpressionTree t)

{

if(t is a leaf)

return value of t's operand ;

else

{

operator = t.element ;

operand1 = evaluate(t.left) ;

operand2 = evaluate(t.right) ;

return(applyOperator(operand1, operator, operand2) ;

}

}

Note:

1. Create Expression tree ADT with the members expression, postfix and value. It has the following functions
   1. void InfixtoPostfix(ExpTree \*t)
   2. void diplay(ExpTree \*t) – displays the infix expression, postfix and the value (initially it will be 0)
   3. void evaluation(Expression \*t)