**SVKM’s NMIMS**

**Mukesh Patel School of Technology Management & Engineering**

**Computer Engineering Department**

Program: B.Tech Integrated Sem V

**Course: Basic Data Structures**

**2024-2025**

Experiment No.01

PART A

(PART A : TO BE REFFERED BY STUDENTS)

**A.1 Aim:**

Introduction to Data Structures and implementation of Arrays

**A.2 Prerequisite:**

1. Knowledge of different types of data structures.

2. Fundamental concepts of C\C++.

**A.3 Outcome:**

**After successful completion of this experiment students will be able to**

1. Identify the need of appropriate selection of data structure
2. Explore the effect of appropriate data structure selection.
3. Differentiate types of data structure based on their organization of data.
4. Enlist the applications of different data structure.
5. Implement arrays for the given problem

**A.4 Theory:**

**A.4.1. Introduction of Data structure**

* The data may be organized into many different ways. The logical and mathematical model of a particular organization of data is called data structure.
* A data structure helps you to understand relationship of one data element with the other and organize it within the memory.
* Data structure specified following:
  + Organization of data
  + Accessing methods
  + Degree of associativity
  + Processing alternatives for information
* Classification of data structure:

A diagram of data structure

Description automatically generated

* Primitive data structure:
  + Basic structures
  + Directly operated upon by the machine instructions
* Non- Primitive data structures:
  + Derived from primitive data structure.
  + Emphasize on structuring of a group of homogenous or heterogeneous data structure.
  + Ex: Arrays, Lists, Files
* Various data structure:

A diagram of a graph

Description automatically generated

**A.5 Procedure/Algorithm:**

**A.5.1:**

**TASK 1:**

Write a C/C++ program of array to perform following **(1D Array)**

1. Find the sum and Average of all the elements in an array.
2. Find highest and lowest element in an array.
3. Write a function that takes an array as input and returns a new array that is the reverse of the original array.

**TASK 2:**

Identify suitable data structure for given scenarios. Specify the reason for it.

1. To implement a system for reversing a word.
2. To implement a printer spooler so that jobs can be printed in the order of the arrival.
3. To represent an image in a form of bitmap.
4. For representing a city region telephone network.
5. To store information about the directories and files in a system.
6. To implement a system for parsing syntax.
7. To implement back functionality in web browsers.
8. To record the sequence of all the pages browsed in one session.
9. To process network packets coming to the router.
10. To represent machines on the internet and to find optimal paths between source machine and destination machine to send data.
11. To represent connections\relations in social networking sites.
12. Google maps to travel from your home to office in minimum time.

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

PART B

(PART B : TO BE COMPLETED BY STUDENTS)

***(Students must submit the soft copy as per following segments within two hours of the practical. The soft copy must be uploaded on the Blackboard or emailed to the concerned lab in charge faculties at the end of the practical in case the there is no Black board access available)***

|  |  |
| --- | --- |
| Roll No. C146 | Name: Manan Gandhi |
| Class : D | Batch : D1 |
| Date of Experiment: 19/07/2024 | Date of Submission |
| Grade : | Time of Submission: |
| Date of Grading: |  |

**B.1 Software Code written by student:**

***(Paste your code completed during the 2 hours of practical in the lab here)***

**Task 1:**

// Write a C/C++ program of array to perform following (1D Array)

// i. Find the sum and Average of all the elements in an array.

// ii. Find highest and lowest element in an array.

// iii. Write a function that takes an array as input and returns a new array that is the reverse of the original array.

#include <iostream>

using namespace std;

int calculate\_sum(int arr[], int len)

{

int sum = 0;

for (int i = 0; i < len; i++)

{

sum += arr[i];

}

return sum;

}

int calculate\_highest(int arr[], int len)

{

int highest = arr[0];

for (int i = 0; i < len; i++)

{

if (arr[i] > highest)

{

highest = arr[i];

}

}

return highest;

}

int calculate\_lowest(int arr[], int len)

{

int lowest = arr[0];

for (int i = 0; i < len; i++)

{

if (arr[i] < lowest)

{

lowest = arr[i];

}

}

return lowest;

}

void reverse\_arr(int arr[], int len, int \*reverse)

{

for (int i = 0; i < len; i++)

{

reverse[i] = arr[len - i - 1];

}

}

int main()

{

int arr[] = {};

double sum = 0, highest, lowest;

int reverse[] = {};

double average = 0;

int len = 0;

cout << "Enter length of array: ";

cin >> len;

for (int i = 0; i < len; i++)

{

cout << "Enter element at index " << i << ": ";

cin >> arr[i];

}

sum = calculate\_sum(arr, len);

average = sum / len;

cout << "Sum: " << sum << endl;

cout << "Average: " << average << endl;

highest = calculate\_highest(arr, len);

lowest = calculate\_lowest(arr, len);

cout << "Highest: " << highest << endl;

cout << "Lowest: " << lowest << endl;

reverse\_arr(arr, len, reverse);

cout << "Reverse: " << endl;

for (int i = 0; i < len; i++)

{

cout << reverse[i] << endl;

}

return 0;

}

**Task 2:**

1. Stack
2. Queue
3. Array
4. Graph
5. Tree
6. Tree
7. Stack
8. Linked List
9. Queue
10. Graph
11. Graph
12. Graph

**Extra Practice Questions:**

**1.**

#include <iostream>

using namespace std;

int main()

{

int a, b, sum;

cout << "Enter a: ";

cin >> a;

cout << "Enter b: ";

cin >> b;

sum = a + b;

cout << "Sum: " << sum << endl;

return 0;

}

**2.**

#include <iostream>

using namespace std;

int main()

{

int nums[] = {};

int len = 0;

cout << "Enter length of array: ";

cin >> len;

for (int i = 0; i < len; i++)

{

cout << "Enter element at index " << i << ": ";

cin >> nums[i];

}

for (int i = 0; i < len; i++)

{

cout << i << ": " << nums[i] << endl;

}

return 0;

}

**3.**

#include <iostream>

#include <unordered\_map>

using namespace std;

int main()

{

int nums[] = {};

int len = 0;

unordered\_map<int, int> frequency;

cout << "Enter length of array: ";

cin >> len;

for (int i = 0; i < len; i++)

{

cout << "Enter element at index " << i << ": ";

cin >> nums[i];

}

for (int i = 0; i < len; i++)

{

frequency[nums[i]] += 1;

}

for (auto x : frequency)

{

cout << x.first << ": " << x.second << endl;

}

return 0;

}

**4.**

#include <iostream>

using namespace std;

int main()

{

int arr[] = {};

int len = 0, num = 0;

cout << "Enter length of array: ";

cin >> len;

for (int i = 0; i < len; i++)

{

cout << "Enter element at index " << i << ": ";

cin >> arr[i];

}

for (int i = 0; i < len; i++)

{

num = (num \* 10) + arr[i];

}

cout << num << endl;

return 0;

}

**B.2 Input and Output:**

***(Paste your program input and output in following format, If there is error then paste the specific error in the output part. In case of error with due permission of the faculty extension can be given to submit the error free code with output in due course of time. Students will be graded accordingly.)***

**Task 1:**

**A screen shot of a computer

Description automatically generated**

**B.3 Conclusion:**

*(****Students must write the conclusion as per the attainment of individual outcome listed above and learning/observation noted in section B.3)***

We learned about different data structures

**B.4 Question of Curiosity**

***(To be answered by student based on the practical performed and learning/observations)***

Q1. Why appropriate selection of data structure is important in computer programming?

For optimized time of execution and space occupancy of the software.

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

**SVKM’s NMIMS**

**Mukesh Patel School of Technology Management & Engineering**

**Computer Engineering Department**

Program: B.Tech Integrated Sem V

**Course: Basic Data Structures**

**2024-2025**

**Experiment No.02**

PART A

(PART A : TO BE REFFERED BY STUDENTS)

**A.1 Aim:**

Introduction to Data Structures and implementation of Arrays

**A.2 Prerequisite:**

1. Knowledge of different types of data structures.

2. Fundamental concepts of C\C++.

**A.3 Outcome:**

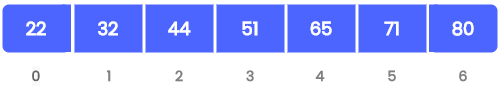
**After successful completion of this experiment students will be able to**

1. Identify the need of appropriate selection of data structure
2. Explore the effect of appropriate data structure selection.
3. Differentiate types of data structure based on their organization of data.
4. Enlist the applications of different data structure.
5. Implement arrays for the given problem

**A.4 Theory:**

### A.4.1. Inserting a new element in an array

A new element can be inserted at any position of an array if there is enough memory space allocated to accommodate the new element. Some of the elements should be shifted forward to keep the order of the elements. Suppose we want to add an element 93 at the 3rd position in the following array.



Initial array

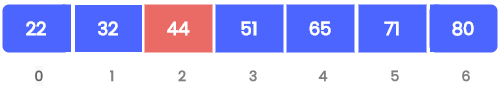
After inserting all elements comes after 32 must be moved to the next location to accommodate the new element and to keep the order of the elements as follows.



Array after inserting a new element

### Deleting an element from an array

Suppose we want to delete element 44 at the 3rd position in the following array.



Initial array

After deletion, all elements coming after 44 must be moved to the previous location to fill the free space and to keep the order of the elements as follows.



Array after deleting an element

#### Algorithm

* Find the position of the element to be deleted in the array.
* If the element is found,  
  Shift all elements after the position of the element by 1 position.  
  Decrement array size by 1.
* If the element is not found: Print “Element Not Found”

**A.5 Procedure/Algorithm:**

**A.5.1:**

**TASK 1:**

Write a C++ program of array to perform following **(1D Array)**

1. Insertion operation in an arrays.

a) Insertion at the end of array

b) Insertion at specific position in an array

1. Deletion operation in an array.

A) Deletion from the end of array

B) Deletion from particular position in array

C) Deleting a particular element in an array

**Task 2:**

**Temperature Monitoring**

You are working on a weather monitoring system that records the daily temperatures of a city over a month. You need to perform various operations such as finding the highest and lowest temperatures, calculating the average temperature, and identifying days when the temperature was above a certain threshold.

**Questions**:

1. **Data Storage**:
   * How would you use a 1D array to store the daily temperatures?
   * Why is a 1D array appropriate for this task?
2. **Implementation**:
   * Write a function to find the highest and lowest temperatures in the month.
   * Write a function to calculate the average temperature for the month.
   * Write a function to count the number of days the temperature was above a given threshold.
3. **Edge Cases**:
   * How would your functions handle an empty array (e.g., no temperature data available)?
   * What would be the output if all temperatures are the same?

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

PART B

(PART B : TO BE COMPLETED BY STUDENTS)

***(Students must submit the soft copy as per following segments within two hours of the practical. The soft copy must be uploaded on the Blackboard or emailed to the concerned lab in charge faculties at the end of the practical in case the there is no Black board access available)***

|  |  |
| --- | --- |
| Roll No. C146 | Name: Manan Gandhi |
| Class : D | Batch : D1 |
| Date of Experiment: 2/7/24 | Date of Submission |
| Grade : | Time of Submission: |
| Date of Grading: |  |

**B.1 Software Code written by student:**

***(Paste your code completed during the 2 hours of practical in the lab here)***

**Task 1:**

/\*

Insertion at specific position in an array

\*/

#include <iostream>

using namespace std;

int main()

{

int arr[] = {1};

int n = 0, index = 0, value = 0;

cout << "Enter length: ";

cin >> n;

cout << "Enter elements of array: " << endl;

for (int i = 0; i < n; i++)

{

cout << i + 1 << ": ";

cin >> arr[i];

}

cout << "Enter index to insert: ";

cin >> index;

cout << "Enter value to insert: ";

cin >> value;

for (int i = n; i >= index; i--)

{

arr[i] = arr[i - 1];

}

arr[index - 1] = value;

cout << "The array is: " << endl;

for (int i = 0; i < n + 1; i++)

{

cout << i + 1 << ": " << arr[i] << endl;

}

}

**Task 2:**

/\*

Insertion at end of array

\*/

#include <iostream>

using namespace std;

int main()

{

int arr[] = {1};

int n = 0, value = 0;

cout << "Enter length: ";

cin >> n;

cout << "Enter elements of array: " << endl;

for (int i = 0; i < n; i++)

{

cout << i + 1 << ": ";

cin >> arr[i];

}

cout << "Enter value to insert: ";

cin >> value;

arr[n] = value;

cout << "The array is: " << endl;

for (int i = 0; i < n + 1; i++)

{

cout << i + 1 << ": " << arr[i] << endl;

}

}

**Task 3:**

/\*

Deletion from the end of array

\*/

#include <iostream>

using namespace std;

int main()

{

int n;

cout << "Enter the size of the array: ";

cin >> n;

int arr[n];

cout << "Enter the elements of the array: ";

for (int i = 0; i < n; i++)

{

cin >> arr[i];

}

n--;

cout << "Array after deletion: ";

for (int i = 0; i < n; i++)

{

cout << arr[i] << " ";

}

cout << endl;

return 0;

}

**Task 4:**

/\*

Deletion from particular position in array

\*/

#include <iostream>

using namespace std;

int main()

{

int n;

cout << "Enter the size of the array: ";

cin >> n;

int arr[n];

cout << "Enter the elements of the array: ";

for (int i = 0; i < n; i++)

{

cin >> arr[i];

}

int pos;

cout << "Enter the position of the element to be deleted: ";

cin >> pos;

for (int i = pos; i < n - 1; i++)

{

arr[i] = arr[i + 1];

}

n--;

cout << "Array after deletion: ";

for (int i = 0; i < n; i++)

{

cout << arr[i] << " ";

}

cout << endl;

return 0;

}

**Task 5:**

/\*

Deleting a particular element in an array

\*/

#include <iostream>

using namespace std;

int main()

{

int n;

cout << "Enter the size of the array: ";

cin >> n;

int arr[n];

cout << "Enter the elements of the array: ";

for (int i = 0; i < n; i++)

{

cin >> arr[i];

}

int element;

cout << "Enter the element to be deleted: ";

cin >> element;

int found = 0;

for (int i = 0; i < n; i++)

{

if (arr[i] == element)

{

found = 1;

for (int j = i; j < n - 1; j++)

{

arr[j] = arr[j + 1];

}

n--;

break;

}

}

if (found == 0)

{

cout << "Element not found in the array" << endl;

}

else

{

cout << "Array after deletion: ";

for (int i = 0; i < n; i++)

{

cout << arr[i] << " ";

}

cout << endl;

}

return 0;

}

**B.2 Input and Output:**

***(Paste your program input and output in following format, If there is error then paste the specific error in the output part. In case of error with due permission of the faculty extension can be given to submit the error free code with output in due course of time. Students will be graded accordingly.)***

**Task 1:**

**A screen shot of a computer code

Description automatically generated**

**Task 2:**

**A screen shot of a computer program

Description automatically generated**

**Task 3:**

**A blue background with white text

Description automatically generated**

**Task 4:**

**A blue background with white text

Description automatically generated**

**Task 5:**

**A screenshot of a computer screen

Description automatically generated**

**B.3 Conclusion:**

*(****Students must write the conclusion as per the attainment of individual outcome listed above and learning/observation noted in section B.3)***

**B.4 Question of Curiosity**

***(To be answered by student based on the practical performed and learning/observations)***

Q1. How an arrays can be declared in the form of an ADT?

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

**SVKM’s NMIMS**

**Mukesh Patel School of Technology Management & Engineering**

**Computer Engineering Department**

Program: B.Tech Integrated Sem V

**Course: Basic Data Structures**

**2024-2025**

**Experiment No.03**

PART A

(PART A : TO BE REFFERED BY STUDENTS)

A.1 Aim: **To study concepts of Structure, Recursion and Pointers in C.**

**Task 1: In a class, there are six students, each having grades for four subjects. Write a program using a 2D array to calculate (i) the total grades for each student and (ii) the average grade for each subject.**

**Task 2: In a small company there are ﬁve salesmen. Each salesman is supposed to sell three**

**products. Write a program using a 2D array to print (i) the total sales by each salesman**

**and (ii) total sales of each item.**

**Task 3: An industrial plant tracks the energy consumption of five machines across four different shifts. Write a program using a 2D array to find (i) the total energy consumption for each machine and (ii) the total energy consumption for each shift.**

**Task 4: Create a structure that store details of BankAccount with account number, balance and customer name.**

**Task 5: Write a C\C++ program-using concept of structures, to store details of BankAccount with account number, balance and customer name. The program must input details from the user and display the details.**

**Task 6: Modify the program written for task 2. The program must store details for *n* accounts. User gives the value of n. The program must provide a search function that searches details of particular bank account.**

**Task 7: Write a C\C++ program with recursive functions to perform following**

**a) Find factorial of a number**

**Task 8:**

**Write a program to swap two numbers using concept of pointers.**

A.2 Prerequisite:

Knowledge of Recursion and Pointer

A.3 Outcome:

After successful completion of this experiment students will be able to

1. Demonstrate the use of recursion and pointers

A.4 Theory:

**Recursive Function:**

A recursive function is one which calls itself. This is another complicated idea which you are unlikely to meet frequently. Recursive functions are useful in evaluating certain types of mathematical function. You may also encounter certain dynamic data structures such as linked lists or binary trees. Recursion is a very useful way of creating and accessing these structures.

### Advantages and Disadvantages of Recursion

Recursion is more elegant and requires few variables which make program clean. Recursion can be used to replace complex nesting code by dividing the problem into same problem of its sub-type. On other hand,memory usage of recursion is higher as it uses stack memory.

Pointer is variable that contains memory address of another variable. From the definition we can say that the pointer is a variable and you can assign different values of the pointer variable. The value contained by the pointer must be an address that indicates the location of another variable in the memory.

**Left Value & Right Value**

For instance, after the integer variable x is declared and assigned to a value like this:

int x;

X=7;

The variable x now has two values:

Left value: 1000

Right value: 7

Where 1000 is the address of the variable x.

**The general form of a pointer declaration is**

data-type \*pointer-name;

**The following shows different types of pointers:**

char \*ptr\_c; /\* declare a pointer to a character \*/

int \*ptr\_int; /\* declare a pointer to an integer \*/

float \*ptr\_flt; /\* declare a pointer to a floating-point \*/

Example 1:

#include <stdio.h>

int main ()

{

int var1;

char var2[10];

printf("Address of var1 variable: %x\n", &var1 );

printf("Address of var2 variable: %x\n", &var2 );

return 0;

}

Output -

Address of var1 variable: bff5a400

Address of var2 variable: bff5a3f6

Example 2:

#include <stdio.h>

int main ()

{

int var = 20; /\* actual variable declaration \*/

int \*ip; /\* pointer variable declaration \*/

ip = &var; /\* store address of var in pointer variable\*/

printf("Address of var variable: %x\n", &var );

/\* address stored in pointer variable \*/

printf("Address stored in ip variable: %x\n", ip );

/\* access the value using the pointer \*/

printf("Value of \*ip variable: %d\n", \*ip );

return 0;

}

Output –

Address of var variable: bffd8b3c

Address stored in ip variable: bffd8b3c

Value of \*ip variable: 20

**NULL Pointers in C**

It is always a good practice to assign a NULL value to a pointer variable in case you do not have exact address to be assigned. This is done at the time of variable declaration. A pointer that is assigned NULL is called a **null** pointer.

The NULL pointer is a constant with a value of zero defined in several standard libraries. Consider the following program:

#include <stdio.h>

int main ()

{

int \*ptr = NULL;

printf("The value of ptr is : %x\n", ptr );

return 0; }

Output –

The value of ptr is 0

PART B

(PART B: TO BE COMPLETED BY STUDENTS)

**(Students must submit the soft copy as per following segments within two hours of the practical. The soft copy must be uploaded on the Portal)**

|  |  |
| --- | --- |
| Roll No. | Name: |
| Program: | Division: |
| Semester: | Batch : |
| Date of Experiment: | Date of Submission: |
| Grade : |  |

Task 1:

Task 2:

Task3:

Task 4:

Task 5:

B.5 Conclusion:

*(****Students must write the conclusion as per the attainment of individual outcome listed above and learning/observation noted in section B.3)***

B.6 Question of Curiosity:

**Write a C\C++ program with recursive functions to perform following**

**a) Find the greatest common divisor (GCD) of two integers.**

**b) Print Fibonacci series up to n term**

**SVKM’s NMIMS**

**Mukesh Patel School of Technology Management & Engineering**

**Computer Engineering Department**

Program: B.Tech Integrated Sem V

**Course: Basic Data Structures**

**2024-2025**

**Experiment No.04**

**A.1 Aim:**

To study and implement concept of Stack data structure.

**A.2 Prerequisite:**

1. Knowledge of different operations performed on Stack data structure

2. Fundamental concepts of C\C++.

**A.3 Outcome:**

**After successful completion of this experiment students will be able to**

1. Identify the need of appropriate selection of data structure
2. Identify the steps of stack data structure selection.
3. Implement stack data structure to solve the given problem
4. Enlist the applications of stack data structure.

**A.4 Theory:**

**A.4.1. Introduction of Stack**

Stack is a linear data structure which follows a particular order in which the operations are performed. The order may be LIFO (Last in first out) or FILO (First in last out).

The functions performed on stack are:

1. **Push:** Adds an item in the stack. If the stack is full, then it is said to be an overflow condition.
2. **Pop:** Removes an item from the stack. The items are popped in the reversed order in which they are pushed. If the stack is empty, then it is said to be an underflow condition.
3. **Peek:** Returns the top most element from the stack.

**Applications of Stack:**

1. Balancing of symbols
2. Infix to postfix\prefix conversion
3. Redo-undo features at many places like editors, photoshop
4. Forward and backward feature in web browsers

**A.5 Procedure/Algorithm:**

**A.5.1:**

**TASK 1:**

Write a program to implement ADT of Stack.

**TASK 2:**

The organization structure of “ABC Pvt. LTD” is given as per joining order. The company is in crisis and wants to start lay off. The employee who joined last is fired first. Implement the suitable data structure and display the order in which the company will fire employees.

|  |  |  |
| --- | --- | --- |
| Employee id | Order of joining |  |
| 101 | 1 | (joined first) |
| 102 | 2 |  |
| 107 | 3 |  |
| 108 | 4 |  |
| 111 | 5 |  |
| 112 | 6 | (joined last) |

PART B

(PART B : TO BE COMPLETED BY STUDENTS)

***(Students must submit the soft copy as per following segments within two hours of the practical. The soft copy must be uploaded on the Blackboard or emailed to the concerned lab in charge faculties at the end of the practical in case the there is no Black board access available)***

|  |  |
| --- | --- |
| Roll No. C146 | Name: Manan Gandhi |
| Class : D | Batch : D1 |
| Date of Experiment: 9/8/24 | Date of Submission |
| Grade : | Time of Submission: |
| Date of Grading: |  |

**B.1 Software Code written by student: (Task 1)**

***(Paste your code completed during the 2 hours of practical in the lab here)***

**Task1:**

/\*

Basic implementation of a stack

\*/

#include <iostream>

using namespace std;

int top = 0, maxstk, stack[] = {};

void push(int);

int pop();

int peek();

void display();

int main()

{

cout << "Enter maxstk: ";

cin >> maxstk;

int number;

input:

int choice;

cout << "Enter 0 for exit, 1 for push, 2 for pop, 3 for peek, 4 for display: ";

cin >> choice;

switch (choice)

{

case 0:

cout << "Final stack is: " << endl;

display();

return 0;

case 1:

cout << "Enter number: ";

cin >> number;

push(number);

break;

case 2:

number = pop();

cout << "Popped element: " << number << endl;

break;

case 3:

number = peek();

cout << "Element at top: " << number << endl;

break;

case 4:

display();

break;

default:

break;

}

goto input;

}

void push(int n)

{

if (top <= maxstk - 1)

{

stack[++top] = n;

}

else

{

cout << "Overflow" << endl;

}

}

int pop()

{

if (top != 0)

{

return stack[top--];

}

else

{

cout << "Underflow" << endl;

return -1;

}

}

int peek()

{

if (top != 0)

{

return stack[top];

}

else

{

cout << "Underflow" << endl;

return -1;

}

}

void display()

{

for (int i = 1; i <= top; i++)

{

cout << stack[i] << " ";

}

cout << endl;

}

**Task2:**

#include <iostream>

using namespace std;

int top = 0, maxstk, stack[] = {};

void push(int);

int pop();

int main()

{

cout << "Enter maxstk: ";

cin >> maxstk;

int number;

for (int i = 0; i < maxstk; i++)

{

int emp;

cout << "Enter employee number: ";

cin >> emp;

push(emp);

}

for (int i = 0; i < maxstk; i++)

{

int emp = pop();

cout << "Emp: " << emp << endl;

}

return 0;

}

void push(int n)

{

if (top <= maxstk - 1)

{

stack[++top] = n;

}

else

{

cout << "Overflow" << endl;

}

}

int pop()

{

if (top != 0)

{

return stack[top--];

}

else

{

cout << "Underflow" << endl;

return -1;

}

}

**Task 3:**

/\*

Pair of brackets

\*/

#include <iostream>

using namespace std;

int open\_top = 0, close\_top = 0, maxstk;

char open[] = {}, close[] = {};

void open\_push(char);

int open\_pop();

void close\_push(char);

int close\_pop();

int main()

{

char brackets[] = {};

cout << "Enter brackets: ";

cin >> brackets;

int i = 0;

while (brackets[i] != '\0')

{

i++;

}

maxstk = i / 2;

i = 0;

while (i < maxstk \* 2)

{

if (brackets[i] == '{' || brackets[i] == '(' || brackets[i] == '[')

{

open\_push(brackets[i]);

}

else

{

close\_push(brackets[i]);

}

i++;

}

bool invalid = false;

for (int i = 0; i < maxstk; i++)

{

char open = open\_pop();

char close = close\_pop();

if (!(open == '(' && close == ')') && !(open == '[' && close == ']') && !(open == '{' && close == '}'))

{

invalid = true;

}

}

if (invalid)

{

cout << "Invalid brackets";

}

else

{

cout << "Valid";

}

cout << endl;

return 0;

}

void open\_push(char c)

{

if (open\_top <= maxstk - 1)

{

open[++open\_top] = c;

}

else

{

cout << "Overflow" << endl;

}

}

int open\_pop()

{

if (open\_top != 0)

{

return open[open\_top--];

}

else

{

cout << "Underflow" << endl;

return -1;

}

}

void close\_push(char c)

{

if (close\_top <= maxstk - 1)

{

close[++close\_top] = c;

}

else

{

cout << "Overflow" << endl;

}

}

int close\_pop()

{

if (close\_top != 0)

{

return close[close\_top--];

}

else

{

cout << "Underflow" << endl;

return -1;

}

}

**B.2 Input and Output: (Task 1)**

***(Paste your program input and output in following format, If there is error then paste the specific error in the output part. In case of error with due permission of the faculty extension can be given to submit the error free code with output in due course of time. Students will be graded accordingly.)***

**Task1:**

**A screenshot of a computer

Description automatically generated**

**Task2:**

**A screenshot of a computer

Description automatically generated**

**Task 3:**

**A blue background with white text

Description automatically generated**

**A blue background with white text

Description automatically generated**

**B.3 Observations and learning [w.r.t. all tasks]:**

***(Students are expected to comment on the output obtained with clear observations and learning for each task/ sub part assigned)***

**B.4 Conclusion:**

*(****Students must write the conclusion as per the attainment of individual outcome listed above and learning/observation noted in section B.3)***

**B.5 Question of Curiosity**

***(To be answered by student based on the practical performed and learning/observations)***

**Checking for palindrome string**

Suppose characters are arriving on a stream reader. Suggest an algorithm to see if the string forms a palindrome. Capitalization, spacing, and punctuation are ignored.

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

**SVKM’s NMIMS**

**Mukesh Patel School of Technology Management & Engineering**

**Computer Engineering Department**

Program: B.Tech Integrated Sem V

**Course: Basic Data Structures**

**2024-2025**

**Experiment No.05**

**A.1 Aim:**

To study and implement concept of Stack data structure.

**A.2 Prerequisite:**

1. Knowledge of different operations performed on Stack data structure

2. Fundamental concepts of C\C++.

**A.3 Outcome:**

**After successful completion of this experiment students will be able to**

1. Identify the need of appropriate selection of data structure
2. Identify the steps of stack data structure selection.
3. Implement stack data structure to solve the given problem
4. Enlist the applications of stack data structure.

**A.4 Theory:**

**A.4.1. Introduction of Stack**

Stack is a linear data structure which follows a particular order in which the operations are performed. The order may be LIFO (Last in first out) or FILO (First in last out).

The functions performed on stack are:

1. **Push:** Adds an item in the stack. If the stack is full, then it is said to be an overflow condition.
2. **Pop:** Removes an item from the stack. The items are popped in the reversed order in which they are pushed. If the stack is empty, then it is said to be an underflow condition.
3. **Peek:** Returns the top most element from the stack.

**Applications of Stack:**

1. Balancing of symbols
2. Infix to postfix\prefix conversion
3. Redo-undo features at many places like editors, photoshop
4. Forward and backward feature in web browsers

**A.5 Procedure/Algorithm:**

**A.5.1:**

**TASK 1:**

Write a program to implement ADT of Stack.

**TASK 2:**

The organization structure of “ABC Pvt. LTD” is given as per joining order. The company is in crisis and wants to start lay off. The employee who joined last is fired first. Implement the suitable data structure and display the order in which the company will fire employees.

|  |  |  |
| --- | --- | --- |
| Employee id | Order of joining |  |
| 101 | 1 | (joined first) |
| 102 | 2 |  |
| 107 | 3 |  |
| 108 | 4 |  |
| 111 | 5 |  |
| 112 | 6 | (joined last) |

PART B

(PART B : TO BE COMPLETED BY STUDENTS)

***(Students must submit the soft copy as per following segments within two hours of the practical. The soft copy must be uploaded on the Blackboard or emailed to the concerned lab in charge faculties at the end of the practical in case the there is no Black board access available)***

|  |  |
| --- | --- |
| Roll No. C146 | Name: Manan Gandhi |
| Class : D | Batch : D1 |
| Date of Experiment: 9/8/24 | Date of Submission |
| Grade : | Time of Submission: |
| Date of Grading: |  |

**B.1 Software Code written by student: (Task 1)**

***(Paste your code completed during the 2 hours of practical in the lab here)***

**Task1:**

*/\**

*Implement the ADT of a stack using structures*

*\*/*

*#include* <iostream>

*using* *namespace* std;

struct *elem*

{

int *val*;

};

void *push*(int);

int *pop*();

int *peek*();

void *display*();

int *maxstk* *=* *50*;

int *top* *=* *-1*;

*elem* *stack*[] *=* {};

int *main*()

{

int *number*;

input:

int *choice*;

*cout* *<<* "Enter 0 for exit, 1 for push, 2 for pop, 3 for peek, 4 for display: ";

*cin* *>>* *choice*;

*switch* (*choice*)

{

*case* *0*:

*cout* *<<* "Final stack is: " *<<* *endl*;

*display*();

*return* *0*;

*case* *1*:

*cout* *<<* "Enter number: ";

*cin* *>>* *number*;

*push*(*number*);

*break*;

*case* *2*:

*number* *=* *pop*();

*cout* *<<* "Popped element: " *<<* *number* *<<* *endl*;

*break*;

*case* *3*:

*number* *=* *peek*();

*cout* *<<* "Element at top: " *<<* *number* *<<* *endl*;

*break*;

*case* *4*:

*display*();

*break*;

*default*:

*break*;

}

*goto* input;

}

void *push*(int *val*)

{

*if* (*top* *<* *maxstk* *-* *1*)

{

*top++*;

*stack*[*top*].*val* *=* *val*;

}

*else*

{

*cout* *<<* "Overflow" *<<* *endl*;

}

}

int *pop*()

{

*if* (*top* *!=* *-1*)

{

int *val* *=* *stack*[*top*].*val*;

*top--*;

*return* *val*;

}

*else*

{

*cout* *<<* "Underflow" *<<* *endl*;

*return* *-1*;

}

}

int *peek*()

{

*if* (*top* *!=* *-1*)

{

*return* *stack*[*top*].*val*;

}

*else*

{

*cout* *<<* "Underflow" *<<* *endl*;

*return* *-1*;

}

}

void *display*()

{

*for* (int *i* *=* *0*; *i* *<=* *top*; *i++*)

{

*cout* *<<* *stack*[*i*].*val* *<<* "*\t*";

}

*cout* *<<* *endl*;

}

**Task2:**

*/\**

*Check whether pair of brackets is valid or not*

*\*/*

*#include* <iostream>

*using* *namespace* std;

struct *elem*

{

char *val*;

};

int *maxstk* *=* *50*;

int *open\_top* *=* *-1*;

int *close\_top* *=* *-1*;

*elem* *open*[] *=* {};

*elem* *close*[] *=* {};

void *open\_push*(char);

char *open\_pop*();

void *close\_push*(char);

char *close\_pop*();

int *main*()

{

char *brackets*[] *=* {};

char *open\_i*, *close\_i*;

*cout* *<<* "Enter brackets: ";

*cin* *>>* *brackets*;

int *i* *=* *0*;

*while* (*brackets*[*i*] *!=* '*\0*')

{

*i++*;

}

*for* (int *j* *=* *0*; *j* *<* *i*; *j++*)

{

*if* (*brackets*[*j*] *==* '(' *||* *brackets*[*j*] *==* '{' *||* *brackets*[*j*] *==* '[')

{

*open\_push*(*brackets*[*j*]);

}

*else* *if* (*brackets*[*j*] *==* ')' *||* *brackets*[*j*] *==* '}' *||* *brackets*[*j*] *==* ']')

{

*close\_push*(*brackets*[*j*]);

}

}

*if* (*open\_top* *==* *close\_top*)

{

*for* (int *j* *=* *0*; *j* *<=* *open\_top*; *j++*)

{

*open\_i* *=* *open\_pop*();

*close\_i* *=* *close\_pop*();

*cout* *<<* *open\_i* *<<* " " *<<* *close\_i* *<<* *endl*;

*if* (*open\_i* *==* '(' *&&* *close\_i* *!=* ')')

{

*cout* *<<* "Invalid" *<<* *endl*;

*return* *0*;

}

*else* *if* (*open\_i* *==* '{' *&&* *close\_i* *!=* '}')

{

*cout* *<<* "Invalid" *<<* *endl*;

*return* *0*;

}

*else* *if* (*open\_i* *==* '[' *&&* *close\_i* *!=* ']')

{

*cout* *<<* "Invalid" *<<* *endl*;

*return* *0*;

}

}

*cout* *<<* "Valid" *<<* *endl*;

}

*else*

{

*cout* *<<* "Invalid" *<<* *endl*;

}

*return* *0*;

}

void *open\_push*(char *val*)

{

*if* (*open\_top* *<* *maxstk* *-* *1*)

{

*open\_top++*;

*open*[*open\_top*].*val* *=* *val*;

}

*else*

{

*cout* *<<* "Overflow" *<<* *endl*;

}

}

char *open\_pop*()

{

*if* (*open\_top* *!=* *-1*)

{

int *val* *=* *open*[*open\_top*].*val*;

*open\_top--*;

*return* *val*;

}

*else*

{

*cout* *<<* "Underflow" *<<* *endl*;

*return* '*\0*';

}

}

void *close\_push*(char *val*)

{

*if* (*close\_top* *<* *maxstk* *-* *1*)

{

*close\_top++*;

*close*[*close\_top*].*val* *=* *val*;

}

*else*

{

*cout* *<<* "Overflow" *<<* *endl*;

}

}

char *close\_pop*()

{

*if* (*close\_top* *!=* *-1*)

{

int *val* *=* *close*[*close\_top*].*val*;

*close\_top--*;

*return* *val*;

}

*else*

{

*cout* *<<* "Underflow" *<<* *endl*;

*return* '*\0*';

}

}

**B.2 Input and Output: (Task 1)**

***(Paste your program input and output in following format, If there is error then paste the specific error in the output part. In case of error with due permission of the faculty extension can be given to submit the error free code with output in due course of time. Students will be graded accordingly.)***

**Task1:**

**A screenshot of a computer

Description automatically generated**

**Task2:**

**A blue background with white text

Description automatically generated**

**A blue background with white text

Description automatically generated**

**B.3 Observations and learning [w.r.t. all tasks]:**

***(Students are expected to comment on the output obtained with clear observations and learning for each task/ sub part assigned)***

**B.4 Conclusion:**

*(****Students must write the conclusion as per the attainment of individual outcome listed above and learning/observation noted in section B.3)***

**B.5 Question of Curiosity**

***(To be answered by student based on the practical performed and learning/observations)***

**Checking for palindrome string**

Suppose characters are arriving on a stream reader. Suggest an algorithm to see if the string forms a palindrome. Capitalization, spacing, and punctuation are ignored.

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

**SVKM’s NMIMS**

**Mukesh Patel School of Technology Management & Engineering**

**Computer Engineering Department**

Program: B.Tech Integrated. Sem V

**Course: Basic Data Structures**

**Experiment No.06**

PART A

(PART A : TO BE REFFERED BY STUDENTS)

**A.1 Aim:**

To study and implement concept of circular queue data structure.

TASK 1: Write a Program to implement a circular Queue

TASK 2: Call Center Management System

Scenario: A call center receives calls from customers, and each call is placed in a waiting queue if all agents are busy. The system has a limited number of slots in the queue to hold these calls. When an agent becomes available, the next call in line is connected. If the queue is full and a new call arrives, the system can either reject the call or replace the oldest one.

Problem:

* The call queue has a fixed size due to system limitations.
* Calls must be handled in the order they arrive (FIFO).
* The system must efficiently manage the fixed number of slots in the queue.

**A.2 Prerequisite:**

1. Knowledge of different operations performed on Queue data structure

2. Fundamental concepts of C\C++.

**A.3 Outcome:**

**After successful completion of this experiment students will be able to**

1. Identify the steps of queue data structure selection.
2. Implement Queue data structure to solve the given problem
3. Enlist the applications of stack data structure.
4. Differentiate between different types of Queue.

**A.4 Theory:**

**A.4.1. Introduction of Queue**

**A.5 Procedure/Algorithm:**

**A.5.1:**

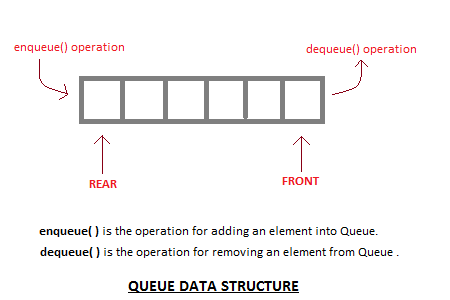
Queue is a linear data structure in which items are inserted at one end called ‘Rear’ and deleted from the other end called ‘Front’. Queues are based on the First-In-First-Out (FIFO) principle that means the data item that is inserted first in the queue is also the first one to be removed from the queue.

A queue or FIFO (first in, first out) is an abstract data type that serves as a collection of elements, with two principal operations: enqueue, the process of adding an element to the collection.(The element is added from the rear side) and dequeue, the process of removing the first element that was added. (The element is removed from the front side). It can be implemented by using both array and linked list.

**Representation of Queue:**

1. Static representation using arrays.
2. Dynamic representation using linked list.

Array representation of a queue needs two indices: FRONT and REAR.



Array declaration in C++ as follows: int queue[maxsize]

**Conditions to be assumed:**

1. FRONT =REAR or FRONT=REAR=-1, if the queue empty
2. Whenever an element is deleted from the queue, FRONT=FRONT+1
3. Whenever an element is inserted into the queue, REAR=REAR+1

**Operations on Queue:**

|  |  |  |
| --- | --- | --- |
| Operation | Description | Restriction |
| Create Queue | It creates a new empty queue. This operation must be carried out in order to make the queue logically accessible | - |
| Qinsert | It inserts a new data item at the rear of the queue | Queue must not be full |
| Qdelete | It deletes and then return the data item at the front of the queue. | Queue must not be empty |
| Queue full | It returns true if the queue is full. Otherwise it returns false. | - |
| Queue Empty | It returns true if the queue is empty. Otherwise it returns false. |  |

PART B

(PART B : TO BE COMPLETED BY STUDENTS)

***(Students must submit the soft copy as per following segments within two hours of the practical. The soft copy must be uploaded on the Blackboard or emailed to the concerned lab in charge faculties at the end of the practical in case the there is no Black board access available)***

|  |  |
| --- | --- |
| Roll No. C146 | Name: Manan Gandhi |
| Class : D | Batch : D1 |
| Date of Experiment: | Date of Submission |
| Grade : | Time of Submission: |
| Date of Grading: |  |

**B.1 Software Code written by student: (Task 1)**

***(Paste your code completed during the 2 hours of practical in the lab here)***

**Task1:**

*// Circular Queue*

*#include* <iostream>

*using* *namespace* std;

*const* int *n* *=* *10*;

int *front* *=* *-1*, *rear* *=* *-1*;

struct *QueueElement*

{

int *value*;

};

*QueueElement* *queue*[*n*];

void *enqueue*(int);

void *dequeue*();

void *display*();

int *main*()

{

int *choice* *=* *0*;

bool *running* *=* *true*;

int *value*;

*while* (*running*)

{

*cout* *<<* *endl*

*<<* "1. Insert" *<<* *endl*

*<<* "2. Delete" *<<* *endl*

*<<* "3. Display" *<<* *endl*

*<<* "0. Exit" *<<* *endl*

*<<* "Enter choice: ";

*cin* *>>* *choice*;

*switch* (*choice*)

{

*case* *1*:

*cout* *<<* "Enter value: ";

*cin* *>>* *value*;

*enqueue*(*value*);

*break*;

*case* *2*:

*dequeue*();

*break*;

*case* *3*:

*display*();

*break*;

*case* *0*:

*running* *=* *false*;

*break*;

*default*:

*cout* *<<* "Invalid choice" *<<* *endl*;

*break*;

}

}

*return* *0*;

}

void *enqueue*(int *value*)

{

*if* ((*front* *==* *0* *&&* *rear* *==* *n* *-* *1*) *||* (*rear* *==* (*front* *-* *1*) *%* (*n* *-* *1*)))

{

*cout* *<<* "Overflow" *<<* *endl*;

*return*;

}

*if* (*front* *==* *-1*)

{

*front* *=* *0*;

*rear* *=* *0*;

}

*else* *if* (*rear* *==* *n* *-* *1* *&&* *front* *!=* *0*)

{

*rear* *=* *0*;

}

*else*

{

*rear++*;

}

*queue*[*rear*].*value* *=* *value*;

}

void *dequeue*()

{

*if* (*front* *==* *-1*)

{

*cout* *<<* "Underflow" *<<* *endl*;

*return*;

}

int *value* *=* *queue*[*front*].*value*;

*if* (*front* *==* *rear*)

{

*front* *=* *-1*;

*rear* *=* *-1*;

}

*else* *if* (*front* *==* *n* *-* *1*)

{

*front* *=* *0*;

}

*else*

{

*front++*;

}

*cout* *<<* "Deleted value: " *<<* *value* *<<* *endl*;

}

void *display*()

{

*if* (*front* *==* *-1* *&&* *rear* *==* *-1*)

{

*cout* *<<* "Empty" *<<* *endl*;

*return*;

}

*if* (*front* *<=* *rear*)

{

*for* (int *i* *=* *front*; *i* *<=* *rear*; *i++*)

{

*cout* *<<* *queue*[*i*].*value* *<<* "*\t*";

}

}

*else*

{

*for* (int *i* *=* *front*; *i* *<* *n*; *i++*)

{

*cout* *<<* *queue*[*i*].*value* *<<* "*\t*";

}

*for* (int *i* *=* *0*; *i* *<=* *rear*; *i++*)

{

*cout* *<<* *queue*[*i*].*value* *<<* "*\t*";

}

}

*cout* *<<* *endl*;

}

**Task2:**

*/\**

*Call Center Management System*

*Scenario: A call center receives calls from customers, and each call is placed in a waiting queue if all agents are busy. The system has a limited number of slots in the queue to hold these calls. When an agent becomes available, the next call in line is connected. If the queue is full and a new call arrives, the system can either reject the call or replace the oldest one.*

*Problem:*

*\* The call queue has a fixed size due to system limitations.*

*\* Calls must be handled in the order they arrive (FIFO).*

*\* The system must efficiently manage the fixed number of slots in the queue.*

*\*/*

*#include* <iostream>

*using* *namespace* std;

*const* int *n* *=* *10*;

int *front* *=* *-1*, *rear* *=* *-1*;

struct *QueueElement*

{

int *value*;

};

*QueueElement* *queue*[*n*];

void *enqueue*(int);

void *dequeue*();

void *display*();

int *main*()

{

int *choice* *=* *0*;

bool *running* *=* *true*;

int *value*;

*while* (*running*)

{

*cout* *<<* *endl*

*<<* "1. Insert" *<<* *endl*

*<<* "2. Delete" *<<* *endl*

*<<* "3. Display" *<<* *endl*

*<<* "0. Exit" *<<* *endl*

*<<* "Enter choice: ";

*cin* *>>* *choice*;

*switch* (*choice*)

{

*case* *1*:

*cout* *<<* "Enter value: ";

*cin* *>>* *value*;

*enqueue*(*value*);

*break*;

*case* *2*:

*dequeue*();

*break*;

*case* *3*:

*display*();

*break*;

*case* *0*:

*running* *=* *false*;

*break*;

*default*:

*cout* *<<* "Invalid choice" *<<* *endl*;

*break*;

}

}

*return* *0*;

}

void *enqueue*(int *value*)

{

*if* ((*front* *==* *0* *&&* *rear* *==* *n* *-* *1*) *||* (*rear* *==* (*front* *-* *1*) *%* (*n* *-* *1*)))

{

*cout* *<<* "Queue is full. Replace the oldest call? (y/n): ";

char *choice*;

*cin* *>>* *choice*;

*if* (*choice* *==* 'n')

{

*return*;

}

*front* *=* (*front* *+* *1*) *%* *n*;

}

*else* *if* (*front* *==* *-1*)

{

*front* *=* *0*;

*rear* *=* *0*;

}

*else* *if* (*rear* *==* *n* *-* *1* *&&* *front* *!=* *0*)

{

*rear* *=* *0*;

}

*else*

{

*rear++*;

}

*queue*[*rear*].*value* *=* *value*;

}

void *dequeue*()

{

*if* (*front* *==* *-1*)

{

*cout* *<<* "Queue is empty" *<<* *endl*;

*return*;

}

*cout* *<<* "Removed call with value: " *<<* *queue*[*front*].*value* *<<* *endl*;

*if* (*front* *==* *rear*)

{

*front* *=* *-1*;

*rear* *=* *-1*;

}

*else*

{

*front* *=* (*front* *+* *1*) *%* *n*;

}

}

void *display*()

{

*if* (*front* *==* *-1*)

{

*cout* *<<* "Queue is empty" *<<* *endl*;

*return*;

}

*cout* *<<* "Calls in queue: ";

*if* (*rear* *>=* *front*)

{

*for* (int *i* *=* *front*; *i* *<=* *rear*; *i++*)

{

*cout* *<<* *queue*[*i*].*value* *<<* " ";

}

}

*else*

{

*for* (int *i* *=* *front*; *i* *<* *n*; *i++*)

{

*cout* *<<* *queue*[*i*].*value* *<<* " ";

}

*for* (int *i* *=* *0*; *i* *<=* *rear*; *i++*)

{

*cout* *<<* *queue*[*i*].*value* *<<* " ";

}

}

*cout* *<<* *endl*;

}

**B.2 Input and Output: (Task 1)**

***(Paste your program input and output in following format, If there is error then paste the specific error in the output part. In case of error with due permission of the faculty extension can be given to submit the error free code with output in due course of time. Students will be graded accordingly.)***

**Task1:**

**B.3 Observations and learning [w.r.t. all tasks]:**

***(Students are expected to comment on the output obtained with clear observations and learning for each task/ sub part assigned)***

**B.4 Conclusion:**

*(****Students must write the conclusion as per the attainment of individual outcome listed above and learning/observation noted in section B.3)***

**B.5 Question of Curiosity**

***(To be answered by student based on the practical performed and learning/observations)***

**Discuss the limitation of linear queue and elaborate any three applications of queue.**

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

**SVKM’s NMIMS**

**Mukesh Patel School of Technology Management & Engineering**

**Computer Engineering Department**

Program: B.Tech Integrated. Sem V

**Course: Basic Data Structures**

LAB Manual

PART A

(PART A : TO BE REFFERED BY STUDENTS)

**Experiment No.07**

**A.1 Aim:**

To study and implement concept of Linked list data structure.

**A.2 Prerequisite:**

1. Knowledge of different operations performed on linked list data structure.
2. Knowledge of different types of linked list and their applications.
3. Fundamental concepts of C\C++.

**A.3 Outcome:**

**After successful completion of this experiment students will be able to**

1. Identify the need of appropriate selection of data structure
2. Identify the steps of linked list data structure selection.
3. Implement different type of Linked list data structure to solve the given problem
4. Enlist the applications of Linked list data structure.
5. Differentiate between different types of Linked list.

**A.4 Theory:**

**A.4.1. Introduction to Linked List**

**A.5 Procedure/Algorithm:**

**A.5.1:**

What are the drawbacks of using sequential storage to represent stacks and queues? One major drawback is that a fixed amount of storage remains allocated to the stack or queue even when the structure is actually using a smaller amount or possibly no storage at all. Further no more than that fixed amount of storage may be allocated, thus introducing the possibility of overflow.

If the memory is allocated for the variable during the compilation (i.e. before execution) of a program, then it is fixed and cannot be changed. For example, an array A [100] is declared with 100 elements, then the allocated memory is fixed and cannot decrease or increase the SIZE of the array if required. So we have to adopt alternative strategy to allocate memory only when it is required. There is a special data structure called linked list that provides a more flexible storage system and it does not require the use of arrays.

In a sequential representation, the items of a stack or queue are implicitly ordered by the sequential order of storage. Thus, if q.items[x] represents an element of a queue, the next element will be q.items[x+1]. Suppose that the items of a stack or q queue were explicitly ordered, that is, each item contained within itself the address of the next item, such an explicit ordering gives rise to a data structure as below, which is known as **Linear Linked List.**

**Info next Info next Info next Info next**

null

**List**

**node node node node**

Each item in the list is called a node and contains two fields, an ***information*** field and a ***next address*** field. The information field holds the actual element on the list. The next address field contains the address of the next node in the list. Such an address, which is used to access a particular node, is known as a pointer. The entire linked list is accessed from an external pointer ***list that*** points to the first node in the list. The next address field of the last node in the list contains a special value, known as ***null***, which is not a valid address. This ***null pointer*** is used to signal the end of the list.

The list with no nodes on it is called ***empty list*** or the ***null list***. The value to the external pointer list to such list is the null pointer. A list can be initialized to the empty list by the operation list=null.

Below figure shows a typical example of node.

PTR

50 NULL

Data Next

PTR->Data=50

PTR->next=NULL

DATA NULL

DATA 0x80031

DATA 0x80010

0x80017

0x80017 0x80010 0x80031

**Advantages and Disadvantages**

Linked list have many advantages and some of them are:

1. Linked list are dynamic structure. That is they can grow or shrink during the execution of a program.
2. Efficient memory utilization: In linked list representation, memory is not pre-allocated. Memory is allocated whenever it is required. And it is de-allocated when it is not needed.
3. Insertion and deletion are easier and efficient. Linked list provides flexibility in inserting a data item at a specified position and deletion of a data item from the given position.
4. Many complex applications can be easily carried out with the linked list.

Linked list has following disadvantages:

1. More memory: to store an integer number, a node with integer data and address field is allocated. That is more memory space is needed.
2. Access to arbitrary data item is little bit cumbersome and also time consuming.

**Operations on Linked List**

The primitive operations performed on the linked list are as follows:

1. Creation
2. Insertion
3. Deletion
4. Traversing
5. Searching
6. Concatenation

**Types of a Linked list:**

Basically we can divide the linked list into the following three types in the order in which they are arranged.

1. Singly linked list
2. Doubly linked list
3. Circular linked list

**Inserting and removing nodes from a list:**

1. A list is a dynamic structure. The number of nodes on a list may vary dramatically as elements are inserted and removed.
2. Suppose we are given a list of integers, as below and we desire to add the integer 6 to the front of the list.

Info next info next info next

8 null

3

5

list

1. The first step is to obtain a node in which to house the additional integer.
2. If a list is to grow and shrink, there must be some way for obtaining empty nodes to be added onto the list.
3. Unlike an array, a list does not come with pre-supplied set of storage locations into which elements can be placed.
4. Let us assume the existence of a mechanism for obtaining empty nodes. The operation p=getnode(); obtains an empty node and sets the contents of a variable named p to the address of that node.
5. The value of p is then a pointer to this newly allocated node.

Info next

p

Info next info next info next

8 null

3

5

list

8. The next step is to insert the integer 6 into the info portion of the newly allocated node. This is done by the operation info(p)=6;

**TASK 1:**

Write a C/C++ program to implement the following operations on a singly linked list

1. Create a linked list (predefined nodes)
2. Create a Linked list with user input of number of nodes

PART B

(PART B : TO BE COMPLETED BY STUDENTS)

***(Students must submit the soft copy as per following segments within two hours of the practical. The soft copy must be uploaded on the Blackboard or emailed to the concerned lab in charge faculties at the end of the practical in case the there is no Black board access available)***

|  |  |
| --- | --- |
| Roll No. C146 | Name: Manan Gandhi |
| Class : D | Batch : D1 |
| Date of Experiment: | Date of Submission |
| Grade : | Time of Submission: |
| Date of Grading: |  |

**B.1 Software Code written by student: (Task 1)**

***(Paste your code completed during the 2 hours of practical in the lab here)***

**Task1:**

*// Linked List*

*#include* <iostream>

*using* *namespace* std;

struct *Node*

{

    int *value*;

*Node* *\*next*;

};

*Node* *\*head* *=* *new* *Node*();

int *main*()

{

*Node* *\*n1* *=* *new* *Node*();

*Node* *\*n2* *=* *new* *Node*();

*Node* *\*n3* *=* *new* *Node*();

*n1*->*value* *=* *10*;

*n2*->*value* *=* *20*;

*n3*->*value* *=* *30*;

*head*->*next* *=* *n1*;

*n1*->*next* *=* *n2*;

*n2*->*next* *=* *n3*;

*Node* *\*current* *=* *head*;

*while* (*current* *!=* *NULL*)

    {

*cout* *<<* *current*->*value* *<<* "*\t*";

*current* *=* *current*->*next*;

    }

*cout* *<<* *endl*;

*return* *0*;

}

**Task2:**

*// Linked List*

*#include* <iostream>

*using* *namespace* std;

struct *Node*

{

    int *value*;

*Node* *\*next*;

};

int *main*()

{

*Node* *\*head* *=* *new* *Node*();

    int *n*;

*cout* *<<* "Enter n: ";

*cin* *>>* *n*;

*Node* *\*current* *=* *head*;

*for* (int *i* *=* *0*; *i* *<* *n*; *i++*)

    {

*Node* *\*n* *=* *new* *Node*();

*n*->*value* *=* *i*;

*n*->*next* *=* *head*;

*head* *=* *n*;

    }

*current* *=* *head*;

*while* (*current* *!=* *NULL*)

    {

*cout* *<<* *current*->*value* *<<* "*\t*";

*current* *=* *current*->*next*;

    }

*cout* *<<* *endl*;

*return* *0*;

}

**B.2 Input and Output: (Task 1)**

***(Paste your program input and output in following format, If there is error then paste the specific error in the output part. In case of error with due permission of the faculty extension can be given to submit the error free code with output in due course of time. Students will be graded accordingly.)***

**Task1:**

**Task2:**

**B.3 Observations and learning [w.r.t. all tasks]:**

***(Students are expected to comment on the output obtained with clear observations and learning for each task/ sub part assigned)***

**B.4 Conclusion:**

*(****Students must write the conclusion as per the attainment of individual outcome listed above and learning/observation noted in section B.3)***

**B.5 Question of Curiosity**

***(To be answered by student based on the practical performed and learning/observations)***

**State the advantages of linked list**

**SVKM’s NMIMS**

**Mukesh Patel School of Technology Management & Engineering**

**Computer Engineering Department**

Program: B.Tech Integrated. Sem V

**Course: Basic Data Structures**

LAB Manual

PART A

(PART A : TO BE REFFERED BY STUDENTS)

**Experiment No.07**

**A.1 Aim:**

To study and implement concept of Linked list data structure.

**A.2 Prerequisite:**

1. Knowledge of different operations performed on linked list data structure.
2. Knowledge of different types of linked list and their applications.
3. Fundamental concepts of C\C++.

**A.3 Outcome:**

**After successful completion of this experiment students will be able to**

1. Identify the need of appropriate selection of data structure
2. Identify the steps of linked list data structure selection.
3. Implement different type of Linked list data structure to solve the given problem
4. Enlist the applications of Linked list data structure.
5. Differentiate between different types of Linked list.

**A.4 Theory:**

**A.4.1. Introduction to Linked List**

**A.5 Procedure/Algorithm:**

**A.5.1:**

What are the drawbacks of using sequential storage to represent stacks and queues? One major drawback is that a fixed amount of storage remains allocated to the stack or queue even when the structure is actually using a smaller amount or possibly no storage at all. Further no more than that fixed amount of storage may be allocated, thus introducing the possibility of overflow.

If the memory is allocated for the variable during the compilation (i.e. before execution) of a program, then it is fixed and cannot be changed. For example, an array A [100] is declared with 100 elements, then the allocated memory is fixed and cannot decrease or increase the SIZE of the array if required. So we have to adopt alternative strategy to allocate memory only when it is required. There is a special data structure called linked list that provides a more flexible storage system and it does not require the use of arrays.

In a sequential representation, the items of a stack or queue are implicitly ordered by the sequential order of storage. Thus, if q.items[x] represents an element of a queue, the next element will be q.items[x+1]. Suppose that the items of a stack or q queue were explicitly ordered, that is, each item contained within itself the address of the next item, such an explicit ordering gives rise to a data structure as below, which is known as **Linear Linked List.**

**Info next Info next Info next Info next**

null

**List**

**node node node node**

Each item in the list is called a node and contains two fields, an ***information*** field and a ***next address*** field. The information field holds the actual element on the list. The next address field contains the address of the next node in the list. Such an address, which is used to access a particular node, is known as a pointer. The entire linked list is accessed from an external pointer ***list that*** points to the first node in the list. The next address field of the last node in the list contains a special value, known as ***null***, which is not a valid address. This ***null pointer*** is used to signal the end of the list.

The list with no nodes on it is called ***empty list*** or the ***null list***. The value to the external pointer list to such list is the null pointer. A list can be initialized to the empty list by the operation list=null.

Below figure shows a typical example of node.

PTR

50 NULL

Data Next

PTR->Data=50

PTR->next=NULL

DATA NULL

DATA 0x80031

DATA 0x80010

0x80017

0x80017 0x80010 0x80031

**Advantages and Disadvantages**

Linked list have many advantages and some of them are:

1. Linked list are dynamic structure. That is they can grow or shrink during the execution of a program.
2. Efficient memory utilization: In linked list representation, memory is not pre-allocated. Memory is allocated whenever it is required. And it is de-allocated when it is not needed.
3. Insertion and deletion are easier and efficient. Linked list provides flexibility in inserting a data item at a specified position and deletion of a data item from the given position.
4. Many complex applications can be easily carried out with the linked list.

Linked list has following disadvantages:

1. More memory: to store an integer number, a node with integer data and address field is allocated. That is more memory space is needed.
2. Access to arbitrary data item is little bit cumbersome and also time consuming.

**Operations on Linked List**

The primitive operations performed on the linked list are as follows:

1. Creation
2. Insertion
3. Deletion
4. Traversing
5. Searching
6. Concatenation

**Types of a Linked list:**

Basically we can divide the linked list into the following three types in the order in which they are arranged.

1. Singly linked list
2. Doubly linked list
3. Circular linked list

**Inserting and removing nodes from a list:**

1. A list is a dynamic structure. The number of nodes on a list may vary dramatically as elements are inserted and removed.
2. Suppose we are given a list of integers, as below and we desire to add the integer 6 to the front of the list.

Info next info next info next

8 null

3

5

list

1. The first step is to obtain a node in which to house the additional integer.
2. If a list is to grow and shrink, there must be some way for obtaining empty nodes to be added onto the list.
3. Unlike an array, a list does not come with pre-supplied set of storage locations into which elements can be placed.
4. Let us assume the existence of a mechanism for obtaining empty nodes. The operation p=getnode(); obtains an empty node and sets the contents of a variable named p to the address of that node.
5. The value of p is then a pointer to this newly allocated node.

Info next

p

Info next info next info next

8 null

3

5

list

8. The next step is to insert the integer 6 into the info portion of the newly allocated node. This is done by the operation info(p)=6;

**TASK 1:**

Write a C/C++ program to implement the following operations on a singly linked list

1. Insert the node at a particular position
2. Modify the info of a node (position to be specified by the user)
3. Delete a node (position to be specified by the user- End, Beginning, position)
4. Traverse the list

PART B

(PART B : TO BE COMPLETED BY STUDENTS)

***(Students must submit the soft copy as per following segments within two hours of the practical. The soft copy must be uploaded on the Blackboard or emailed to the concerned lab in charge faculties at the end of the practical in case the there is no Black board access available)***

|  |  |
| --- | --- |
| Roll No. | Name: |
| Class : | Batch : |
| Date of Experiment: | Date of Submission |
| Grade : | Time of Submission: |
| Date of Grading: |  |

**B.1 Software Code written by student: (Task 1)**

***(Paste your code completed during the 2 hours of practical in the lab here)***

**Task1:**

/\*

Write a C/C++ program to implement the following operations on a singly linked list

i) Insert the node at a particular position

ii) Modify the info of a node (position to be specified by the user)

iii) Delete a node (position to be specified by the user- End, Beginning, position)

iv) Traverse the list

\*/

#include <iostream>

using namespace std;

struct Node

{

int data;

Node \*next;

};

Node \*head = NULL;

void insertNode(int data, int position)

{

Node \*newNode = new Node();

newNode->data = data;

newNode->next = NULL;

if (position == 1)

{

newNode->next = head;

head = newNode;

return;

}

Node \*temp = head;

for (int i = 0; i < position - 2; i++)

{

temp = temp->next;

}

newNode->next = temp->next;

temp->next = newNode;

}

void modifyNode(int data, int position)

{

Node \*temp = head;

for (int i = 0; i < position - 1; i++)

{

temp = temp->next;

}

temp->data = data;

}

void deleteNode(int position)

{

Node \*temp = head;

if (position == 1)

{

head = temp->next;

delete temp;

return;

}

for (int i = 0; i < position - 2; i++)

{

temp = temp->next;

}

Node \*temp2 = temp->next;

temp->next = temp2->next;

delete temp2;

}

void traverseList()

{

Node \*temp = head;

while (temp != NULL)

{

cout << temp->data << " ";

temp = temp->next;

}

cout << endl;

}

int main()

{

int choice;

do

{

cout << "1. Insert Node\n2. Modify Node\n3. Delete Node\n4. Traverse List\n5. Exit\n";

cout << "Enter your choice: ";

cin >> choice;

switch (choice)

{

case 1:

int data, position;

cout << "Enter data and position: ";

cin >> data >> position;

insertNode(data, position);

break;

case 2:

cout << "Enter data and position: ";

cin >> data >> position;

modifyNode(data, position);

break;

case 3:

cout << "Enter position: ";

cin >> position;

deleteNode(position);

break;

case 4:

traverseList();

break;

case 5:

break;

default:

cout << "Invalid choice\n";

}

} while (choice != 5);

return 0;

}

**B.2 Input and Output: (Task 1)**

***(Paste your program input and output in following format, If there is error then paste the specific error in the output part. In case of error with due permission of the faculty extension can be given to submit the error free code with output in due course of time. Students will be graded accordingly.)***

**Task1:**

A screenshot of a computer program

Description automatically generated

**B.3 Observations and learning [w.r.t. all tasks]:**

***(Students are expected to comment on the output obtained with clear observations and learning for each task/ sub part assigned)***

**B.4 Conclusion:**

*(****Students must write the conclusion as per the attainment of individual outcome listed above and learning/observation noted in section B.3)***

**B.5 Question of Curiosity**

***(To be answered by student based on the practical performed and learning/observations)***

**State the advantages of linked list**

**SVKM’s NMIMS**

**Mukesh Patel School of Technology Management & Engineering**

**Computer Engineering Department**

Program: B.Tech Integrated. Sem V

**Course: Basic Data Structures**

LAB Manual

PART A

(PART A : TO BE REFFERED BY STUDENTS)

**Experiment No.09**

**A.1 Aim:**

To study and implement STACK and QUEUE using Linked List.

**A.2 Prerequisite:**

1. Knowledge of different operations performed on linked list data structure.
2. Knowledge of different types of linked list and their applications.
3. Fundamental concepts of C\C++.

**A.3 Outcome:**

**After successful completion of this experiment students will be able to**

1. Identify the need of appropriate selection of data structure
2. Identify the steps of linked list data structure selection.
3. Implement different type of Linked list data structure to solve the given problem
4. Enlist the applications of Linked list data structure.
5. Differentiate between different types of Linked list.

**A.4 Theory:**

**A.4.1. Introduction to Linked List**

**A.5 Procedure/Algorithm:**

**A.5.1:**

What are the drawbacks of using sequential storage to represent stacks and queues? One major drawback is that a fixed amount of storage remains allocated to the stack or queue even when the structure is actually using a smaller amount or possibly no storage at all. Further no more than that fixed amount of storage may be allocated, thus introducing the possibility of overflow.

If the memory is allocated for the variable during the compilation (i.e. before execution) of a program, then it is fixed and cannot be changed. For example, an array A [100] is declared with 100 elements, then the allocated memory is fixed and cannot decrease or increase the SIZE of the array if required. So we have to adopt alternative strategy to allocate memory only when it is required. There is a special data structure called linked list that provides a more flexible storage system and it does not require the use of arrays.

In a sequential representation, the items of a stack or queue are implicitly ordered by the sequential order of storage. Thus, if q.items[x] represents an element of a queue, the next element will be q.items[x+1]. Suppose that the items of a stack or q queue were explicitly ordered, that is, each item contained within itself the address of the next item, such an explicit ordering gives rise to a data structure as below, which is known as **Linear Linked List.**

**Info next Info next Info next Info next**

null

**List**

**node node node node**

Each item in the list is called a node and contains two fields, an ***information*** field and a ***next address*** field. The information field holds the actual element on the list. The next address field contains the address of the next node in the list. Such an address, which is used to access a particular node, is known as a pointer. The entire linked list is accessed from an external pointer ***list that*** points to the first node in the list. The next address field of the last node in the list contains a special value, known as ***null***, which is not a valid address. This ***null pointer*** is used to signal the end of the list.

The list with no nodes on it is called ***empty list*** or the ***null list***. The value to the external pointer list to such list is the null pointer. A list can be initialized to the empty list by the operation list=null.

Below figure shows a typical example of node.

PTR

50 NULL

Data Next

PTR->Data=50

PTR->next=NULL

DATA NULL

DATA 0x80031

DATA 0x80010

0x80017

0x80017 0x80010 0x80031

**Advantages and Disadvantages**

Linked list have many advantages and some of them are:

1. Linked list are dynamic structure. That is they can grow or shrink during the execution of a program.
2. Efficient memory utilization: In linked list representation, memory is not pre-allocated. Memory is allocated whenever it is required. And it is de-allocated when it is not needed.
3. Insertion and deletion are easier and efficient. Linked list provides flexibility in inserting a data item at a specified position and deletion of a data item from the given position.
4. Many complex applications can be easily carried out with the linked list.

Linked list has following disadvantages:

1. More memory: to store an integer number, a node with integer data and address field is allocated. That is more memory space is needed.
2. Access to arbitrary data item is little bit cumbersome and also time consuming.

**Operations on Linked List**

The primitive operations performed on the linked list are as follows:

1. Creation
2. Insertion
3. Deletion
4. Traversing
5. Searching
6. Concatenation

**Types of a Linked list:**

Basically we can divide the linked list into the following three types in the order in which they are arranged.

1. Singly linked list
2. Doubly linked list
3. Circular linked list

**Inserting and removing nodes from a list:**

1. A list is a dynamic structure. The number of nodes on a list may vary dramatically as elements are inserted and removed.
2. Suppose we are given a list of integers, as below and we desire to add the integer 6 to the front of the list.

Info next info next info next

8 null

3

5

list

1. The first step is to obtain a node in which to house the additional integer.
2. If a list is to grow and shrink, there must be some way for obtaining empty nodes to be added onto the list.
3. Unlike an array, a list does not come with pre-supplied set of storage locations into which elements can be placed.
4. Let us assume the existence of a mechanism for obtaining empty nodes. The operation p=getnode(); obtains an empty node and sets the contents of a variable named p to the address of that node.
5. The value of p is then a pointer to this newly allocated node.

Info next

p

Info next info next info next

8 null

3

5

list

8. The next step is to insert the integer 6 into the info portion of the newly allocated node. This is done by the operation info(p)=6;

Refer the ppt for theory

**TASK 1:**

Implement STACK and QUEUE using linked list.

PART B

(PART B : TO BE COMPLETED BY STUDENTS)

***(Students must submit the soft copy as per following segments within two hours of the practical. The soft copy must be uploaded on the Blackboard or emailed to the concerned lab in charge faculties at the end of the practical in case the there is no Black board access available)***

|  |  |
| --- | --- |
| Roll No. C146 | Name: Manan Gandhi |
| Class : D | Batch : D1 |
| Date of Experiment: 4/10/24 | Date of Submission: 4/10/24 |
| Grade : | Time of Submission: |
| Date of Grading: |  |

**B.1 Software Code written by student: (Task 1)**

***(Paste your code completed during the 2 hours of practical in the lab here)***

**Task1:**

*// Stack using linked list*

*#include* <iostream>

*using* *namespace* std;

struct *Node*

{

int *val*;

*Node* *\*next*;

};

*Node* *\*head* *=* *NULL*;

void *push*(int);

int *pop*();

void *display*();

int *main*()

{

int *choice*, *val*;

*while* (*1*)

{

*cout* *<<* "1. Push*\n*2. Pop*\n*3. Display*\n*4. Exit*\n*Enter your choice: ";

*cin* *>>* *choice*;

*switch* (*choice*)

{

*case* *1*:

*cout* *<<* "Enter value to push: ";

*cin* *>>* *val*;

*push*(*val*);

*break*;

*case* *2*:

*val* *=* *pop*();

*if* (*val* *!=* *-1*)

*cout* *<<* "Popped value: " *<<* *val* *<<* *endl*;

*break*;

*case* *3*:

*display*();

*break*;

*case* *4*:

*exit*(*0*);

*default*:

*cout* *<<* "Invalid choice" *<<* *endl*;

}

}

}

void *push*(int *val*)

{

*Node* *\*newNode* *=* *new* *Node*;

*newNode*->*val* *=* *val*;

*newNode*->*next* *=* *head*;

*head* *=* *newNode*;

}

int *pop*()

{

*if* (*head* *==* *NULL*)

{

*cout* *<<* "Stack is empty" *<<* *endl*;

*return* *-1*;

}

*Node* *\*temp* *=* *head*;

*head* *=* *head*->*next*;

int *val* *=* *temp*->*val*;

*delete* *temp*;

*return* *val*;

}

void *display*()

{

*if* (*head* *==* *NULL*)

{

*cout* *<<* "Stack is empty" *<<* *endl*;

*return*;

}

*Node* *\*temp* *=* *head*;

*cout* *<<* "Stack: ";

*while* (*temp* *!=* *NULL*)

{

*cout* *<<* *temp*->*val* *<<* " ";

*temp* *=* *temp*->*next*;

}

*cout* *<<* *endl*;

}

**Task2:**

*// Queue using linked list*

*#include* <iostream>

*using* *namespace* std;

struct *Node*

{

int *val*;

*Node* *\*next*;

};

*Node* *\*head* *=* *NULL*;

void *enqueue*(int);

int *dequeue*();

void *display*();

int *main*()

{

int *choice*, *val*;

*while* (*1*)

{

*cout* *<<* "1. Enqueue*\n*2. Dequeue*\n*3. Display*\n*4. Exit*\n*Enter your choice: ";

*cin* *>>* *choice*;

*switch* (*choice*)

{

*case* *1*:

*cout* *<<* "Enter value to enqueue: ";

*cin* *>>* *val*;

*enqueue*(*val*);

*break*;

*case* *2*:

*val* *=* *dequeue*();

*if* (*val* *!=* *-1*)

*cout* *<<* "Dequeued value: " *<<* *val* *<<* *endl*;

*break*;

*case* *3*:

*display*();

*break*;

*case* *4*:

*exit*(*0*);

*default*:

*cout* *<<* "Invalid choice" *<<* *endl*;

}

}

}

void *enqueue*(int *val*)

{

*Node* *\*newNode* *=* *new* *Node*;

*newNode*->*val* *=* *val*;

*newNode*->*next* *=* *NULL*;

*if* (*head* *==* *NULL*)

{

*head* *=* *newNode*;

}

*else*

{

*Node* *\*temp* *=* *head*;

*while* (*temp*->*next* *!=* *NULL*)

{

*temp* *=* *temp*->*next*;

}

*temp*->*next* *=* *newNode*;

}

}

int *dequeue*()

{

*if* (*head* *==* *NULL*)

{

*cout* *<<* "Queue is empty" *<<* *endl*;

*return* *-1*;

}

*Node* *\*temp* *=* *head*;

*head* *=* *head*->*next*;

int *val* *=* *temp*->*val*;

*delete* *temp*;

*return* *val*;

}

void *display*()

{

*if* (*head* *==* *NULL*)

{

*cout* *<<* "Queue is empty" *<<* *endl*;

*return*;

}

*Node* *\*temp* *=* *head*;

*cout* *<<* "Queue: ";

*while* (*temp* *!=* *NULL*)

{

*cout* *<<* *temp*->*val* *<<* " ";

*temp* *=* *temp*->*next*;

}

*cout* *<<* *endl*;

}

**B.2 Input and Output: (Task 1)**

***(Paste your program input and output in following format, If there is error then paste the specific error in the output part. In case of error with due permission of the faculty extension can be given to submit the error free code with output in due course of time. Students will be graded accordingly.)***

**Task1:**

**A screenshot of a computer program

Description automatically generated**

**Task2:**

**A screenshot of a computer program

Description automatically generated**

**B.3 Observations and learning [w.r.t. all tasks]:**

***(Students are expected to comment on the output obtained with clear observations and learning for each task/ sub part assigned)***

**B.4 Conclusion:**

*(****Students must write the conclusion as per the attainment of individual outcome listed above and learning/observation noted in section B.3)***

**B.5 Question of Curiosity**

***(To be answered by student based on the practical performed and learning/observations)***

**State the advantages of linked list**