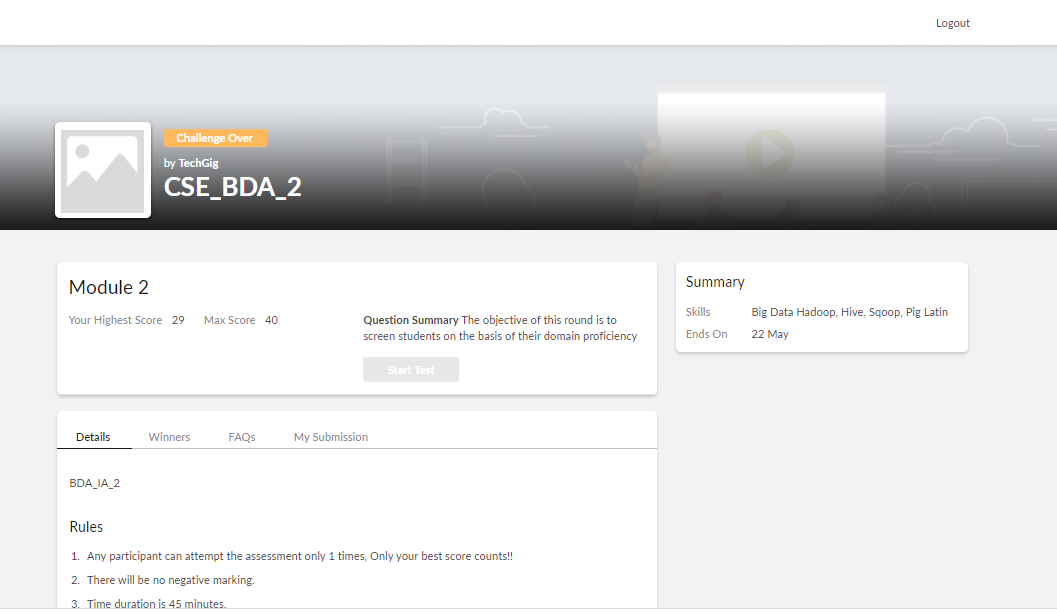
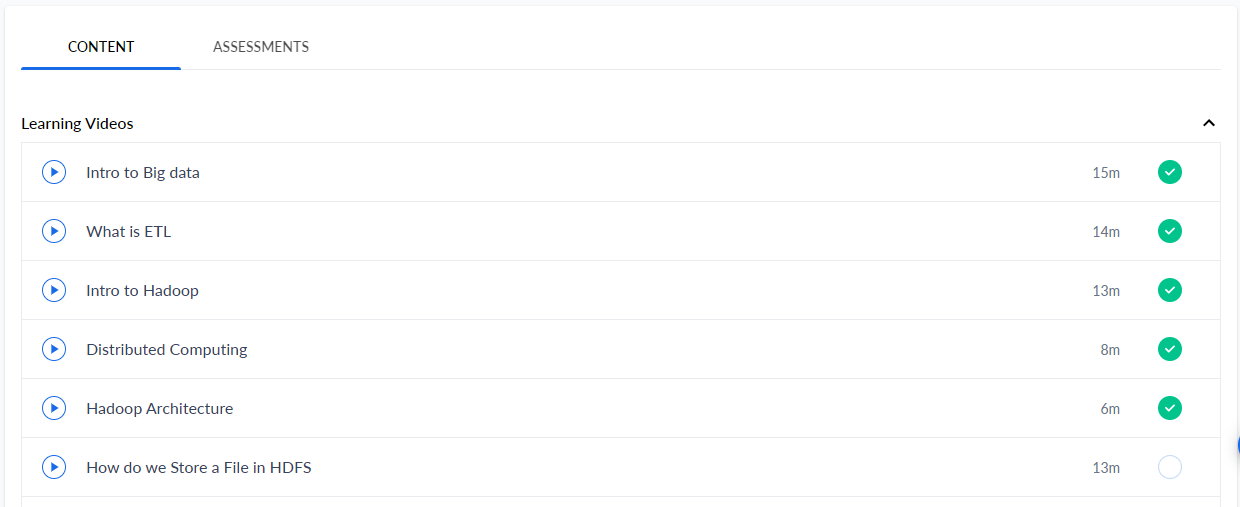
**DAILY ONLINE ACTIVITIES SUMMARY**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Date:** | **22-05-2020** | | | | | **Name:** | **Anusha** | |
| **Sem & Sec** | **VIII Semester & A Section** | | | | | **USN:** | **4AL16CS014** | |
| **Online Test Summary** | | | | | | | | |
| **Subject** | | **Big Data Analytics** | | | | | | |
| **Max. Marks** | | **40** | | **Score** | | | **29** | |
| **Certification Course Summary** | | | | | | | | |
| **Course** | **Introduction to Hadoop** | | | | | | | |
| **Certificate Provider** | | | **Great Learning** | | **Duration** | | | **One video(6 mins)** |
| **Coding Challenges** | | | | | | | | |
| **Problem Statement:** performing operations on singly linked list stack **.** | | | | | | | | |
| **Status: COMPLETED** | | | | | | | | |
| **Uploaded the report in Github** | | | | | **YES** | | | |
| **If yes Repository name** | | | | | **anushasuvarna-014** | | | |
| **Uploaded the report in slack** | | | | | **YES** | | | |

Online Test Details:



Certification Course Details:



Hadoop Architecture:

Hadoop has three major components:

1. HDFS
2. MapReduce
3. YARN

In Hadoop Distributed File System there are two nodes namenode and datanode. Here namenode acts like a master and datanode acts like a slave we can have any number of slaves. In HDFS we can delete any file but small editing is little difficult but is possible by using other platform.HDFS takes care of storage.

Coding Challenges Details:

Write a C program to perform operations on singly linked list stack

|  |
| --- |
|  |
|  |
|  |  |
|  |  |
|  | **Solution:** |
|  | |  | | --- | | #include <stdio.h> | |  | #include <stdlib.h> | |  |  | |  | struct node | |  | { | |  | int info; | |  | struct node \*ptr; | |  | }\*top,\*top1,\*temp; | |  |  | |  | int topelement(); | |  | void push(int data); | |  | void pop(); | |  | void empty(); | |  | void display(); | |  | void destroy(); | |  | void stack\_count(); | |  | void create(); | |  |  | |  | int count = 0; | |  |  | |  | void main() | |  | { | |  | int no, ch, e; | |  |  | |  | printf("\n 1 - Push"); | |  | printf("\n 2 - Pop"); | |  | printf("\n 3 - Top"); | |  | printf("\n 4 - Empty"); | |  | printf("\n 5 - Exit"); | |  | printf("\n 6 - Dipslay"); | |  | printf("\n 7 - Stack Count"); | |  | printf("\n 8 - Destroy stack"); | |  |  | |  | create(); | |  |  | |  | while (1) | |  | { | |  | printf("\n Enter choice : "); | |  | scanf("%d", &ch); | |  |  | |  | switch (ch) | |  | { | |  | case 1: | |  | printf("Enter data : "); | |  | scanf("%d", &no); | |  | push(no); | |  | break; | |  | case 2: | |  | pop(); | |  | break; | |  | case 3: | |  | if (top == NULL) | |  | printf("No elements in stack"); | |  | else | |  | { | |  | e = topelement(); | |  | printf("\n Top element : %d", e); | |  | } | |  | break; | |  | case 4: | |  | empty(); | |  | break; | |  | case 5: | |  | exit(0); | |  | case 6: | |  | display(); | |  | break; | |  | case 7: | |  | stack\_count(); | |  | break; | |  | case 8: | |  | destroy(); | |  | break; | |  | default : | |  | printf(" Wrong choice, Please enter correct choice "); | |  | break; | |  | } | |  | } | |  | } | |  |  | |  | /\* Create empty stack \*/ | |  | void create() | |  | { | |  | top = NULL; | |  | } | |  |  | |  | /\* Count stack elements \*/ | |  | void stack\_count() | |  | { | |  | printf("\n No. of elements in stack : %d", count); | |  | } | |  |  | |  | /\* Push data into stack \*/ | |  | void push(int data) | |  | { | |  | if (top == NULL) | |  | { | |  | top =(struct node )malloc(1sizeof(struct node)); | |  | top->ptr = NULL; | |  | top->info = data; | |  | } | |  | else | |  | { | |  | temp =(struct node )malloc(1sizeof(struct node)); | |  | temp->ptr = top; | |  | temp->info = data; | |  | top = temp; | |  | } | |  | count++; | |  | } | |  |  | |  | /\* Display stack elements \*/ | |  | void display() | |  | { | |  | top1 = top; | |  |  | |  | if (top1 == NULL) | |  | { | |  | printf("Stack is empty"); | |  | return; | |  | } | |  |  | |  | while (top1 != NULL) | |  | { | |  | printf("%d ", top1->info); | |  | top1 = top1->ptr; | |  | } | |  | } | |  |  | |  | /\* Pop Operation on stack \*/ | |  | void pop() | |  | { | |  | top1 = top; | |  |  | |  | if (top1 == NULL) | |  | { | |  | printf("\n Error : Trying to pop from empty stack"); | |  | return; | |  | } | |  | else | |  | top1 = top1->ptr; | |  | printf("\n Popped value : %d", top->info); | |  | free(top); | |  | top = top1; | |  | count--; | |  | } | |  |  | |  | /\* Return top element \*/ | |  | int topelement() | |  | { | |  | return(top->info); | |  | } | |  |  | |  | /\* Check if stack is empty or not \*/ | |  | void empty() | |  | { | |  | if (top == NULL) | |  | printf("\n Stack is empty"); | |  | else | |  | printf("\n Stack is not empty with %d elements", count); | |  | } | |  |  | |  | /\* Destroy entire stack \*/ | |  | void destroy() | |  | { | |  | top1 = top; | |  |  | |  | while (top1 != NULL) | |  | { | |  | top1 = top->ptr; | |  | free(top); | |  | top = top1; | |  | top1 = top1->ptr; | |  | } | |  | free(top1); | |  | top = NULL; | |  |  | |  | printf("\n All stack elements destroyed"); | |  | count = 0; | |  | } | |  |  | |  | @SWATHI829 | |  | SWATHI829 1 hour ago | |  | #include <stdio.h> | |  | #include <stdlib.h> | |  |  | |  | struct node | |  | { | |  | int info; | |  | struct node \*ptr; | |  | }\*top,\*top1,\*temp; | |  |  | |  | int topelement(); | |  | void push(int data); | |  | void pop(); | |  | void empty(); | |  | void display(); | |  | void destroy(); | |  | void stack\_count(); | |  | void create(); | |  |  | |  | int count = 0; | |  |  | |  | void main() | |  | { | |  | int no, ch, e; | |  |  | |  | printf("\n 1 - Push"); | |  | printf("\n 2 - Pop"); | |  | printf("\n 3 - Top"); | |  | printf("\n 4 - Empty"); | |  | printf("\n 5 - Exit"); | |  | printf("\n 6 - Dipslay"); | |  | printf("\n 7 - Stack Count"); | |  | printf("\n 8 - Destroy stack"); | |  |  | |  | create(); | |  |  | |  | while (1) | |  | { | |  | printf("\n Enter choice : "); | |  | scanf("%d", &ch); | |  |  | |  | switch (ch) | |  | { | |  | case 1: | |  | printf("Enter data : "); | |  | scanf("%d", &no); | |  | push(no); | |  | break; | |  | case 2: | |  | pop(); | |  | break; | |  | case 3: | |  | if (top == NULL) | |  | printf("No elements in stack"); | |  | else | |  | { | |  | e = topelement(); | |  | printf("\n Top element : %d", e); | |  | } | |  | break; | |  | case 4: | |  | empty(); | |  | break; | |  | case 5: | |  | exit(0); | |  | case 6: | |  | display(); | |  | break; | |  | case 7: | |  | stack\_count(); | |  | break; | |  | case 8: | |  | destroy(); | |  | break; | |  | default : | |  | printf(" Wrong choice, Please enter correct choice "); | |  | break; | |  | } | |  | } | |  | } | |  |  | |  | /\* Create empty stack \*/ | |  | void create() | |  | { | |  | top = NULL; | |  | } | |  |  | |  | /\* Count stack elements \*/ | |  | void stack\_count() | |  | { | |  | printf("\n No. of elements in stack : %d", count); | |  | } | |  |  | |  | /\* Push data into stack \*/ | |  | void push(int data) | |  | { | |  | if (top == NULL) | |  | { | |  | top =(struct node )malloc(1sizeof(struct node)); | |  | top->ptr = NULL; | |  | top->info = data; | |  | } | |  | else | |  | { | |  | temp =(struct node )malloc(1sizeof(struct node)); | |  | temp->ptr = top; | |  | temp->info = data; | |  | top = temp; | |  | } | |  | count++; | |  | } | |  |  | |  | /\* Display stack elements \*/ | |  | void display() | |  | { | |  | top1 = top; | |  |  | |  | if (top1 == NULL) | |  | { | |  | printf("Stack is empty"); | |  | return; | |  | } | |  |  | |  | while (top1 != NULL) | |  | { | |  | printf("%d ", top1->info); | |  | top1 = top1->ptr; | |  | } | |  | } | |  |  | |  | /\* Pop Operation on stack \*/ | |  | void pop() | |  | { | |  | top1 = top; | |  |  | |  | if (top1 == NULL) | |  | { | |  | printf("\n Error : Trying to pop from empty stack"); | |  | return; | |  | } | |  | else | |  | top1 = top1->ptr; | |  | printf("\n Popped value : %d", top->info); | |  | free(top); | |  | top = top1; | |  | count--; | |  | } | |  |  | |  | /\* Return top element \*/ | |  | int topelement() | |  | { | |  | return(top->info); | |  | } | |  |  | |  | /\* Check if stack is empty or not \*/ | |  | void empty() | |  | { | |  | if (top == NULL) | |  | printf("\n Stack is empty"); | |  | else | |  | printf("\n Stack is not empty with %d elements", count); | |  | } | |  |  | |  | /\* Destroy entire stack \*/ | |  | void destroy() | |  | { | |  | top1 = top; | |  |  | |  | while (top1 != NULL) | |  | { | |  | top1 = top->ptr; | |  | free(top); | |  | top = top1; | |  | top1 = top1->ptr; | |  | } | |  | free(top1); | |  | top = NULL; | |  |  | |  | printf("\n All stack elements destroyed"); | |  | count = 0; | |  |  | |  | @Ashwithpinto | |  | Ashwithpinto 39 minutes ago | |  | #include <stdio.h> | |  | #include <stdlib.h> | |  |  | |  | struct node | |  | { | |  | int info; | |  | struct node \*ptr; | |  | }\*top,\*top1,\*temp; | |  |  | |  | int topelement(); | |  | void push(int data); | |  | void pop(); | |  | void empty(); | |  | void display(); | |  | void destroy(); | |  | void stack\_count(); | |  | void create(); | |  |  | |  | int count = 0; | |  |  | |  | void main() | |  | { | |  | int no, ch, e; | |  |  | |  | printf("\n 1 - Push"); | |  | printf("\n 2 - Pop"); | |  | printf("\n 3 - Top"); | |  | printf("\n 4 - Empty"); | |  | printf("\n 5 - Exit"); | |  | printf("\n 6 - Dipslay"); | |  | printf("\n 7 - Stack Count"); | |  | printf("\n 8 - Destroy stack"); | |  |  | |  | create(); | |  |  | |  | while (1) | |  | { | |  | printf("\n Enter choice : "); | |  | scanf("%d", &ch); | |  |  | |  | switch (ch) | |  | { | |  | case 1: | |  | printf("Enter data : "); | |  | scanf("%d", &no); | |  | push(no); | |  | break; | |  | case 2: | |  | pop(); | |  | break; | |  | case 3: | |  | if (top == NULL) | |  | printf("No elements in stack"); | |  | else | |  | { | |  | e = topelement(); | |  | printf("\n Top element : %d", e); | |  | } | |  | break; | |  | case 4: | |  | empty(); | |  | break; | |  | case 5: | |  | exit(0); | |  | case 6: | |  | display(); | |  | break; | |  | case 7: | |  | stack\_count(); | |  | break; | |  | case 8: | |  | destroy(); | |  | break; | |  | default : | |  | printf(" Wrong choice, Please enter correct choice "); | |  | break; | |  | } | |  | } | |  | } | |  |  | |  | /\* Create empty stack \*/ | |  | void create() | |  | { | |  | top = NULL; | |  | } | |  |  | |  | /\* Count stack elements \*/ | |  | void stack\_count() | |  | { | |  | printf("\n No. of elements in stack : %d", count); | |  | } | |  |  | |  | /\* Push data into stack \*/ | |  | void push(int data) | |  | { | |  | if (top == NULL) | |  | { | |  | top =(struct node )malloc(1sizeof(struct node)); | |  | top->ptr = NULL; | |  | top->info = data; | |  | } | |  | else | |  | { | |  | temp =(struct node )malloc(1sizeof(struct node)); | |  | temp->ptr = top; | |  | temp->info = data; | |  | top = temp; | |  | } | |  | count++; | |  | } | |  |  | |  | /\* Display stack elements \*/ | |  | void display() | |  | { | |  | top1 = top; | |  |  | |  | if (top1 == NULL) | |  | { | |  | printf("Stack is empty"); | |  | return; | |  | } | |  |  | |  | while (top1 != NULL) | |  | { | |  | printf("%d ", top1->info); | |  | top1 = top1->ptr; | |  | } | |  | } | |  |  | |  | /\* Pop Operation on stack \*/ | |  | void pop() | |  | { | |  | top1 = top; | |  |  | |  | if (top1 == NULL) | |  | { | |  | printf("\n Error : Trying to pop from empty stack"); | |  | return; | |  | } | |  | else | |  | top1 = top1->ptr; | |  | printf("\n Popped value : %d", top->info); | |  | free(top); | |  | top = top1; | |  | count--; | |  | } | |  |  | |  | /\* Return top element \*/ | |  | int topelement() | |  | { | |  | return(top->info); | |  | } | |  |  | |  | /\* Check if stack is empty or not \*/ | |  | void empty() | |  | { | |  | if (top == NULL) | |  | printf("\n Stack is empty"); | |  | else | |  | printf("\n Stack is not empty with %d elements", count); | |  | } | |  |  | |  | /\* Destroy entire stack \*/ | |  | void destroy() | |  | { | |  | top1 = top; | |  |  | |  | while (top1 != NULL) | |  | { | |  | top1 = top->ptr; | |  | free(top); | |  | top = top1; | |  | top1 = top1->ptr; | |  | } | |  | free(top1); | |  | top = NULL; | |  |  | |  | printf("\n All stack elements destroyed"); | |  | count = 0; | |