Ramakrishna Mission Vidyamandira

(An Autonomous College Under University of Calcutta)

Topic: Data Structures

Submitted by
Class Roll Number: 261 Name: Rajkumar Das Subject: Computer Science UG-1 B.Sc. 2nd Semester Batch: 2023-27

Q1) Write a program to insert an element in a position in an array.

Code:

```
#include<stdio.h>
int main(){
int arr[100],n,i,pos,elem;
printf("enter the number of elements of the array: \n");
scanf("%d",&n);
printf("enter %d elements\n",n);
for(i = 0;i<n;i++){
scanf("%d",&arr[i]);
}
printf("enter the element to insert and position: \n");
scanf("%d %d",&elem,&pos);
for(i = n-1;i>=pos-1;i--){
arr[i+1] = arr[i];
}
arr[pos-1] = elem;
printf("new array is: ");
for(i = 0;i<=n;i++){
printf("%d ",arr[i]);
}
return 0;
}
```

Output:

enter the number of elements of the array:

5

enter 5 elements

4 1 8 6 3

enter the element to insert and position:

22 4

new array is: 4 1 8 22 6 3

Algorithm:

- Begin the main function.
- Declare variables arr as an array of integers of size 100, n, i, pos, and elem.
- Print a message asking the user to enter the number of elements of the array (n).
- Read the input for n.
- Print a message asking the user to enter n elements of the array.
- Iterate i from 0 to n-1: a. Read the elements of the array from the user and store them in the arr array.
- Print a message asking the user to enter the element to insert and its position (elem and pos).
- Read the input for elem and pos.
- Iterate i from n-1 to pos-1 (inclusive) backwards: a. Shift the elements of the array to the right starting from pos-1 position to make room for the new element.
- Assign the value of elem to arr[pos-1] to insert it at the desired position.
- Print "new array is: ".
- Iterate i from 0 to n (inclusive): a. Print the elements of the array.
- End the main function.

Discussion:

- User Input and Array Initialization:
    - The program starts by prompting the user to input the number of elements (n) they want to insert into the array. Then, it prompts the user to input n elements.
    - It initializes an array arr of size 100 to store these elements. The array size is hardcoded, which may not be optimal as it limits the maximum number of elements the user can input.
- Insertion Process:
    - After taking the elements as input, the program prompts the user to enter the element (elem) they want to insert and the position (pos) at which they want to insert it.
    - To insert the element at the specified position, the program shifts the elements from the specified position to the end of the array one place to the right to make space for the new element. This process is executed using a loop that iterates backward through the array.

- Output:
    - Once the insertion is done, the program prints the updated array, including the newly inserted element.
- Potential Issues:
    - The loop that prints the elements of the array after insertion runs from 0 to n, which might lead to printing garbage values because the array might not be completely filled with valid elements up to index n. It should iterate up to n-1.
    - There's no check for the validity of the position entered by the user. If the user enters a position outside the valid range (less than 1 or greater than n+1), the behavior of the program is undefined.
    - The program doesn't handle the case when the array is already full (n reaches 100). It assumes there's always enough space to insert a new element, which might lead to a buffer overflow if the array is already at maximum capacity.

Q2) Write a program to delete an element from a position in an array.

Code:

```
#include<stdio.h>

int main(){

int arr[10],pos,n;

printf("enter the number of elemetns of the array: \n");

scanf("%d",&n);

printf("enter the elements: \n");

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

scanf("%d",&arr[i]);

printf("enter the position to delete: \n");

scanf("%d",&pos);

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

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

printf("the array is\n");

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

printf("%d ",arr[i]);

return 0;
```

}

Output:

enter the number of elements of the array:

5

enter the elements:

1 2 3 4 5

enter the position to delete:

3

the array is

1 2 4 5

Algorithm:

- Begin the main function.
- Declare variables arr as an array of integers of size 10, pos, and n.
- Print a message asking the user to enter the number of elements of the array (n).
- Read the input for n.
- Print a message asking the user to enter n elements of the array.
- Iterate i from 0 to n-1: a. Read the elements of the array from the user and store them in the arr array.
- Print a message asking the user to enter the position of the element to delete (pos).
- Read the input for pos.
- Iterate i from pos-1 to n-1: a. Shift the elements of the array to the left starting from pos-1 position to overwrite the element to be deleted.
- Print "the array is".
- Iterate i from 0 to n-2: a. Print the elements of the array, excluding the deleted element.
- End the main function.

Discussion:

- User Input and Array Initialization:
    - The program starts by prompting the user to input the number of elements (n) they want to initialize the array with.
    - Then, it asks the user to input n elements, which are stored in the array arr.
    - The array size is fixed to 10 elements, which may limit the number of elements the user

can work with.

- Deletion Process:
  - After taking the elements as input, the program prompts the user to enter the position (pos) from which they want to delete an element.
  - To delete the element at the specified position, the program shifts all elements to the left starting from the specified position to overwrite the element to be deleted. This process is executed using a loop that iterates from the specified position to the end of the array.
- Output:
  - Once the deletion is done, the program prints the updated array without the deleted element.
- Potential Issues:
  - The loop that prints the elements of the array after deletion runs from 0 to n-2, which correctly excludes the deleted element. However, it could lead to an incorrect output if n is 0 or 1 because in those cases, it should either not print anything (for n=0) or print the single remaining element (for n=1).
  - There's no check for the validity of the position entered by the user. If the user enters a position outside the valid range (less than 1 or greater than n), the behavior of the program is undefined.

Q3) Write a program to take sparse matrix as an input, print it and create its triplet.

Code:

```
#include<stdio.h>
int main(){



int row,col,i,j,count = 0;
printf("enter the number of row: \n");
scanf("%d",&row);

printf("enter the number of column: \n");
scanf("%d",&col);

int arr1[row][col];

printf("enter the elements of sparse matrix: \n");
for(i = 0;i<row;i++){
for(j = 0;j <col;j++){
scanf("%d",&arr1[i][j]);
if(arr1[i][j] != 0)
count++;
}
}



int row1 = 1,col1 = 0;
int arr2[count+1][3];
```

```
arr2[0][0] = row; arr2[0][1] = col; arr2[0][2] = count;



for(i = 0;i<row;i++){
for(j = 0;j<col;j++){
if(arr1[i][j] != 0){
arr2[row1][col1] = i;
arr2[row1][col1+1] = j;
arr2[row1][col1+2] = arr1[i][j];
row1++;
}
}
}

printf("the triplet is: \n");
for(i = 0;i<count+1;i++){
for(j = 0;j<3;j++){
printf("%d ",arr2[i][j]);
}
printf("\n");
}
return 0;
}
```

Output:

```
enter the number of row:
4
enter the number of column:
```

3

enter the elements of sparse matrix:

0 1 0

0 0 1

2 0 4

0 0 3

the triplet is:

4 3 5

0 1 1

1 2 1

2 0 2

2 2 4

3 2 3

Algorithm:

- Begin the main function.
- Declare variables row, col, i, j, and count.
- Prompt the user to enter the number of rows (row) of the matrix and read the input.
- Prompt the user to enter the number of columns (col) of the matrix and read the input.
- Declare a 2D array arr1 with dimensions row × col to store the elements of the matrix.
- Print a message asking the user to enter the elements of the sparse matrix.
- Iterate i from 0 to row-1: a. Iterate j from 0 to col-1: i. Read the elements of the matrix from the user and store them in arr1[i][j]. ii. If the element is non-zero, increment the count variable.
- Declare variables row1 and col1, both initialized to 1.
- Declare a 2D array arr2 with dimensions (count+1) × 3 to store the triplet representation of the sparse matrix.
- Store the number of rows, columns, and count of non-zero elements in the first row of arr2.
- Iterate i from 0 to row-1: a. Iterate j from 0 to col-1: i. If arr1[i][j] is non-zero:- Store the row index (i), column index (j), and the value (arr1[i][j]) in arr2.- Increment row1.
- Print "the triplet is:".
- Iterate i from 0 to count: a. Iterate j from 0 to 2: i. Print the elements of arr2. b. Print a newline character.
- End the main function.

Discussion:

- User Input and Matrix Initialization:
    - The program begins by prompting the user to input the number of rows (row) and columns (col) of the matrix.
    - It then initializes a 2D array arr1 of size row × col to store the elements of the matrix.
    - The user is prompted to input the elements of the matrix, and the code reads and populates arr1 accordingly.
    - While reading the matrix elements, the code also counts the number of non-zero elements (count).
- Triplet Representation:
    - The code then initializes another 2D array arr2 to store the triplet representation of the sparse matrix. The size of arr2 is determined by count + 1 rows (including the first row for storing metadata) and 3 columns.
    - The first row of arr2 is filled with metadata: the number of rows, columns, and the count of non-zero elements.
    - The subsequent rows of arr2 store the triplet representation of the non-zero elements, where each triplet consists of the row index, column index, and the corresponding non-zero value from the matrix.
- Output:
    - After constructing the triplet representation, the code prints out the resulting triplets in a readable format.

Q4) Write a program to create a stack and implement its operation, push , pop , isEmpty , isFull .

Code:

```
#include<stdio.h>

#define S_SIZE 5


void push(int elem,int* stack,int *index);

void pop(int* stack,int *index);

void peek(int* stack,int *index);

void isFull(int *index);

void isEmpty(int *index);

void display(int *stack,int size);


int main(){
```

```
// initilization of the stack;
int index = 0; // index points to the first empty place
int arr[S_SIZE] = {0};
int choice,num;

do{
printf("enter your choice: \n 1-> push .2-> pop\n 3-> peek \n 4-> display stack \n 5-> isEmpty. 6-> isFull \n 7-> exit :\n");
scanf("%d",&choice);
switch (choice){
case 1:
printf("enter element: \n");
scanf("%d",&num);
push(num,arr,&index);
break;
case 2:
pop(arr,&index);
break;
case 3:
peek(arr,&index);
break;
case 4:
display(arr,S_SIZE);
break;
case 5:
isEmpty(&index);
break;
case 6:
isFull(&index);
```

```
break;
case 7:
return 0;
break;
default:
printf("wrong input!!\n");
}
}while(choice!=7);
}

void display(int *stack,int size){
for(int i = 0;i<size;i++){
printf("%d ",*(stack+i));
}
printf("\n");
}

void isEmpty(int *index){
if (*index == 0){
printf("true\n");
}
else{
printf("false\n");
}
}

void isFull(int *index){
if (*index == S_SIZE){
printf("true\n");
```

```
}
else{
printf("false\n");
}
}

void peek(int* stack,int *index){
if(*index == 0){
printf("no element to peek!!\n");
}
else{
printf("the top most element in the stack is %d\n",*(stack+(*index)-1));
}
}

void pop(int *stack,int *index){
if((*index)-1 > -1){
*(stack+(*index)-1) = 0;
(*index)--;
printf("element popped,successfully!!\n");

}
else{
printf("underflow.stack is empty!!\n");
}
}



void push(int elem,int *stack,int *index){
```

```
int i;
for(i = 0;i<S_SIZE;i++){
if(*index<S_SIZE){
if(*(stack+i) == 0){
*index = i;
*(stack+*index) = elem;
(*index)++;
printf("element pushed,successfully!!\n");
break;
}
}
else{
printf("overflow.the stack is full!!\n");
break;
}
}
}
```

Output:

```
enter your choice:
1-> push .2-> pop
3-> peek
4-> display stack
5-> isEmpty. 6-> isFull
7-> exit :
1
enter element:
43
```

element pushed,successfully!!

enter your choice:

1-> push .2-> pop

3-> peek

4-> display stack

5-> isEmpty. 6-> isFull

7-> exit :

1

enter element:

55

element pushed,successfully!!

enter your choice:

1-> push .2-> pop

3-> peek

4-> display stack

5-> isEmpty. 6-> isFull

7-> exit :

1

enter element:

52

element pushed,successfully!!

enter your choice:

1-> push .2-> pop

3-> peek

4-> display stack

5-> isEmpty. 6-> isFull

7-> exit :

1

enter element:

64

element pushed,successfully!!

enter your choice:

1-> push .2-> pop

3-> peek

4-> display stack

5-> isEmpty. 6-> isFull

7-> exit :

1

enter element:

82

element pushed,successfully!!

enter your choice:

1-> push .2-> pop

3-> peek

4-> display stack

5-> isEmpty. 6-> isFull

7-> exit :

4

43 55 52 64 82

enter your choice:

1-> push .2-> pop

3-> peek

4-> display stack

5-> isEmpty. 6-> isFull

7-> exit :

1

enter element:

28

overflow.the stack is full!!

enter your choice:

1-> push .2-> pop

3-> peek

4-> display stack

5-> isEmpty. 6-> isFull

7-> exit :

2

element popped,successfully!!

enter your choice:

1-> push .2-> pop

3-> peek

4-> display stack

5-> isEmpty. 6-> isFull

7-> exit :

2

element popped,successfully!!

enter your choice:

1-> push .2-> pop

3-> peek

4-> display stack

5-> isEmpty. 6-> isFull

7-> exit :

2

element popped,successfully!!

enter your choice:

1-> push .2-> pop

3-> peek

4-> display stack

5-> isEmpty. 6-> isFull

7-> exit :

2

element popped,successfully!!

enter your choice:

1-> push .2-> pop

3-> peek

4-> display stack

5-> isEmpty. 6-> isFull

7-> exit :

2

element popped,successfully!!

enter your choice:

1-> push .2-> pop

3-> peek

4-> display stack

5-> isEmpty. 6-> isFull

7-> exit :

2

underflow.stack is empty!!

enter your choice:

1-> push .2-> pop

3-> peek

4-> display stack

5-> isEmpty. 6-> isFull

7-> exit :

7

Algorithm:

- Begin the main function.
- Declare variables index, arr (the stack array), and choice.
- Initialize index to 0, indicating the first empty place in the stack array.
- Initialize the stack array arr with all elements set to 0.
- Enter a do-while loop until the user chooses to exit (choice 7):
  a. Prompt the user to input their choice (push, pop, peek, display, isEmpty, isFull, or exit).
  b. Read the user's choice.
  c. Switch based on the choice:
  i. Case 1: If the user chooses to push, prompt for the element and call the push function.
  ii. Case 2: If the user chooses to pop, call the pop function.
  iii. Case 3: If the user chooses to peek, call the peek function.
  iv. Case 4: If the user chooses to display the stack, call the display function.
  v. Case 5: If the user chooses to check if the stack is empty, call the isEmpty function.
  vi. Case 6: If the user chooses to check if the stack is full, call the isFull function.
  vii. Case 7: If the user chooses to exit, return 0 and terminate the program.
  viii. Default: Print "wrong input!!".
- End the do-while loop.
- End the main function.
- Functions:
- display: Print the elements of the stack array.
- isEmpty: Check if the stack is empty based on the value of index.
- isFull: Check if the stack is full based on the value of index.
- peek: Print the topmost element of the stack.
- pop: Remove the topmost element of the stack.
- push: Add an element to the stack if it is not full.

Discussion:

- Functionality:
  - The code allows users to perform various operations on the stack, including push, pop, peek, display, check if the stack is empty, and check if the stack is full.
  - The stack is implemented using a simple array, with the index variable indicating the top of the stack.
- User Interaction:
  - The code provides a user-friendly interface where users can choose the operation they want to perform.
  - Users are prompted to input their choice, and appropriate actions are taken based on

their selection.
    - Error messages are displayed for invalid inputs or when attempting to pop from an empty stack.
- Push Operation:
    - The push function adds an element to the stack if there is space available.
    - It iterates through the stack array to find an empty slot and inserts the element at that position.
    - If the stack is full, it displays an overflow message.
- Pop Operation:
    - The pop function removes the topmost element from the stack.
    - It decrements the index variable to simulate removing the element.
    - If the stack is empty, it displays an underflow message.
- Peek Operation:
    - The peek function allows users to view the topmost element of the stack without removing it.
    - It checks if the stack is empty before attempting to peek and displays an appropriate message.
- Display Operation:
    - The display function prints all elements of the stack array.
    - It provides users with a visual representation of the stack's contents.