20/12/23

1. 2nd largest element

AIM:

To find the second largest element in an array

ALGORITHM:-

1. Initialize two variables first largest and second largest to first element of the array

2. Iterate over the array starting from the second element.

   - If current element is greater than first largest, update second largest to value of first largest and update first largest to current element.

   - current element less than first element but greater than second largest and not equal to first largest, update second largest to current element.

3. After the loop, second largest will hold the value of second largest element is equal to first largest element.

PROGRAM -

```
#include<stdio.h>
int first second largest (int arr[], int size)
{
  int first largest = arr[0];
  int second largest = arr[0];
  for (int i=1; i<size; i++)
  {
    if (arr[i] > first largest)
  }
  second largest = first largest;
  first largest = arr[i];
  } else if (arr[i] > second largest && arr[i] != first largest)
  {
  second largest = arr[i];

  }
return second largest;
}
int main(){
    int size;
    printf("Enter the size of the array: ");
    scanf("%d", &size);

int arr[size];
    printf("Enter the element: \n", size);
    for (int i=0; i<size; i++) {
      scanf("%d", &arr[i]);

}
```

int secondlargest = findsecondlargest(arr, size);
printf("the second largest element in
array is : %d\n", secondlargest);
return 0;

OUTPUT:-

Enter the size of the array: 5

Enter 5 elements:

2 3 5 8 9

The second largest element in array is: 8

RESULT:- Ok C

Thus program is successfully implement for find 2nd largest number in given array

28/12/23

2. Odd numbers in odd indices

AIM:-

To find any odd numbers in odd indices of a given integer array.

ALGORITHM:-

1. Start the program.
2. Declare n and i.
3. Ask for n and read it.
4. Declare arr[n]
5. Ask for arr elements and read them
6. Print "odd numbers at odd indices".
7. For i=1 to n with step 2, if arr[i] is odd, print arr[i].
8. End the program.

CODING:-

```
#include<stdio.h>
int main(){
int x, i;
printf("Given the num of elements in array");
scanf("%d", &x);
int arr[x];
printf("Given element of array");
    for (i=0; i<n; i++) {
        scanf("%d", &arr[i]);
}
```

```
printf("odd numbers at odd indices:");
for(i=1;i<n;i+=2){
    if(arr[i]%2!=0){
        printf("%d",arr[i]);
    }
}
return 0;
}
```

OUTPUT:-

Enter the number of elements in the array: 5

Enter the element of the array: 6 7 8 9 10

odd numbers at odd indices: 7 9

RESULT:- The C

Thus program successfully implemented for finding odd numbers in odd indices

21/12/23

3. Linear search

AIM:

To write a c-program for linear search

ALGORITHM:

Step 1:- Let K be the element we are searching for.

Step 2:- Check each element in the list by comparing it to K.

Step 3:- If any element is equal to K, return its index

Step 4:- If we reach the end of the list without finding the element equal to K, return some value to represent that the element is not found.

PROGRAM:

```
#include<stdio.h>
int linearsearch(int *arr, int size, int key)
{
for(int i=0; i<n; i++)
  {
   if (*(arr+i)==key)
   {
    return i;
  }}
```

```
return -1;
}
int main()
{
int n;
printf("Enter the size of the array:");
scanf("%d",&n);
int *arr=(int*)malloc(n*sizeof(int));
printf("Enter the element of the array:");
for(int i=0;i<n;i++)
{
scanf("%d",&(*(arr+i)));
}
int key;
printf("Enter the element to search:");
scanf("%d",&key);
int result=linearsearch(arr,n,key);
if(result!=-1)
{
printf("Element found at index %d",result);
}
else
{
printf("Element not found");
}
free(arr);
return 0;
```

OUTPUT:

Enter the size of the array: 5

Enter the elements of the array 6 7 8 9 10

Enter the element to search: 8

Element found at index: 2

RESULT:-

THE Program is successfully implemented for linear search.

2/12/23

4. Binary search

AIM:- To
To write a program for binary search

ALGORITHM:-

1) Start the program
2) Declare x, arr[n], key and result.
3) Ask for n and read it.
4) Ask for key and read it.
5) Ask for arr elements and read them.
6) call binarysearch (arr,n,key) and store the result.
7) If result is not -1, print "Element found at index: "result". else print "Element not found"
8) End the program

PROGRAM:

```
#include <stdio.h>
int binarysearch (int arr[], int n, int key) {
    int low=0;
    int high=n-1;

while (low<=high) {
        int mid=(low+high)/2;
```

```
if(arr[mid] == key){
        return mid;

  }
else if(arr[mid] < key){
        low = mid+1;

  }
else
  {
  high = mid-1;

  }
}
return -1;

}
int main(){
      int n;
      Printf("Enter the size of the array:");
      scanf("%d", &n);

      int arr[n];
      Printf("Enter the elements of the array:");

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

  }
  int key;
  Printf("Enter the element to search:");

  scanf("%d", &key);

  int result = binary search (arr, n, key);
  if(result != -1){ c

        Printf("Element found at index: %d",
                                    result);
```

```
}
else
    printf("Element not found:");

}
return 0;
}
```

OUTPUT:-

Enter the size of the array: 5

Enter the elements of the array: 4 5 6 7 8

Enter the element to search: 7

Element found at index: 3

RESULT:-

Thus program successfully implemented for binary search.

25/9/23

5. LINKED LIST Implementation of STACK

AIM:

To write a C-Program for linked list implementation of stack.

ALGORITHM:-

1. Start the Program.
2. Declare a struct node with data and next fields.
3. Initialize top to NULL.
4. Define push(data) function:
   - Allocate memory for a new node.
   - Set data and next fields of the new node.
   - Update top to point to the new node.
5. Define pop() function:
   - If top is NULL, return -1;
   - Save top->data in data;
   - Update top to top->next;
   - Free the old top node
   - Return data.

6. Define display() function:

- Iterate over the stack from top to NULL.
- Print data of each node

7. In main(), push 1, 2, 3, and 4 onto the stack

8. Display the stack elements.

9. Pop two elements from the stack and print them.

10. Display the stack elements again.

11. End the program

PROGRAM:-

```
#Include<stdio.h>
#Include<stdlib.h>
struct node{
    int data;
    struct node* next;

};
struct node* top=NULL;
void push(int data){
    struct node* newnode=(struct node*)malloc
                                (sizeof(struct));
```

```
newnode -> data = data;
newnode -> next = top;
top = newnode;
}
int pop(){
    if(top == NULL){
        return -1;
    }
    int data = top -> data;
    top = top -> next;
    free(newnode);
    return data;
}
void display(){
    struct node *ptr = top;
    while(ptr != NULL){
        printf("%d", ptr -> data);
        ptr = ptr -> next;
    }
    printf("\n");
}
int main(){
    push(1);
    push(2);
    push(3);
    push(4);
```

```
printf("stack elements:");
display();
printf("popped element: %d\n", pop());
printf("popped element: %d\n", pop());
printf("stack elements:");
display();
return 0;
}
```

OUTPUT:-

stack elements: 4 3 2 1

popped element: 4

popped element: 3

stack elements: 2 1

RESULT:-

Thus the program is successfully implemented to linked list implementation of stack

22/10/23 6. Infix to postfix

AIM:

To write a C program for implementing infix to postfix conversion.

ALGORITHM:

1. Start the program
2. Declare a stack and a variable top to keep track of the top of the stack.
3. Define a function push(x) to push an element onto the stack.
4. Define a function pop() to pop an element from the stack.
5. Define a function priority(x) to return the priority of an operator.
6. In the main function:
   - Ask user an expression and record it.
   - For each character c in the expression:
     - If c is alphanumeric, print it.
     - If c is "(", push it onto the stack.

- If o is ')', pop and print elements from the stack until '(' is found.
- If o is an operator, pop and print elements from the stack until the top element has equal or higher priority, then push o onto the stack.
- Pop and print all remaining elements from the stack.

7. Find the program.

PROGRAM:-

```
#include<stdio.h>
#include<ctype.h>
char stack[100];
int top=-1;
void push(char x)
{
stack[++top]=x;
}
char pop()
{
if(top==-1)
    return-1;
else
    return stack[top--];
}
```

```
int priority(char x)
{
 if(x=='(')
    return 0;
 if(x=='+'||x=='-')
    return 1;
 if(x=='*'||x=='/')
    return 2;
 return 0;
}
int main()
{
 char exp[100];
 char *e,x;
 printf("Enter the expression:");
 scanf("%s",exp);
 printf("\n")
 e=exp;
 while(*e!='\0')
 {
  if (isalnum(*e))
      printf("%c",*e);
  else if(*e=='(')
      push(*e);
  else if(*e=='*')
```

```
{
while((x=pop())!='(')
    printf("%c",x);

}
else
{
while(priority(stack[top])>=priority('*'))
  printf("%c",pop());
  push('*');
}
ctt;
}
while(top!=-1)
  {
  printf("%c",pop());
  }
return 0;
}
```

OUTPUT:-

Enter the expression:- a+b

a b +

RESULT:-

Thus the program is successfully implemented for converting stack infix to postfix

27/12/23

7. Array implementation using queue

AIM:

C program of array implementation of queue.

ALGORITHM:-

1. Initialize a queue with a front and rear pointer and an array to store elements.

2. Enqueue (insert) an element by incrementing the rear pointer and adding the element to the array at the rear position.

3. Dequeue (remove) an element by incrementing the front (pointer).

4. Check for queue overflow (when the rear pointer exceeds the array size) and underflow (when the front pointer exceeds the rear pointer).

5. Display the elements of the queue.

PROGRAM:-

```
#include<stdio.h>
#include<stdlib.h>
#define MAX 100;
int queue[MAX];
int front=-1;
int rear=-1;
void enqueue(int item)
{
 if (rear==MAX-1)
  {
  printf("queue overflow\n");
  }
else
 {
 rear=(rear+1)%MAX;
 queue[rear]=item;
 }
int dequeue()
{
 if (front==-1 || front > rear)
  {
  printf("queue underflow\n");
  return 0;
  }
int item=queue(front);
front=(front+1)%MAX;
return item;
}
```

```
void display()
  {
   If(front==-1)
     {
      Printf("Queue is empty\n");
     }
   Else
     {
      Printf("Queue is \n");
      for (int i=front; i<=rear; i++)
        {
         Printf("%d", queue[i]);
        }
      Printf("\n");
     }
  }
int main()
  {
   int choice;
   int item;
   Printf("1. Enqueue\n");
   Printf("2. Dequeue\n");
   Printf("3. display\n");
   Printf("4. Exit\n");
```

```
do
{
scanf("%d", &choice);
switch(choice)
{
 case 1:
    printf("Insert the element in queue");
    scanf("%d", &item);
    enqueue(item);
    break;

case 2:
    item = dequeue();
    if (item != 0)
    {
     printf("Element deleted from queue
                                is : %d\n", item);
    }
    break;

case 3:
    display();
    break;

case 4:
    exit(0);
 }
}
while(choice != 4);
 return 0;
};
```

OUTPUT:

1. Enqueue
2. Dequeue
3. Display
4. Exit

Insert the element: 34

Insert the element: 32

Queue underflow

RESULT:

The program is successfully implemented for array implementation using queue

27/12/23

8. Balancing Parenthesis

AIM:-

To write a C program for balancing parentheses.

ALGORITHM:-

1. Initialize an empty stack.
2. Iterate through each character in the input string
3. If the character is an opening parenthesis (i.e. '(', '{', or '[') push it onto the stack.
4. If the character is a closing parenthesis (i.e. ')', '}' or ']'), do the following
   a. If the stack is empty, return false (unbalanced parenthesis)
   b. Pop the top element from the stack.
   c. If the popped element is not the corresponding opening parenthesis for current closing parenthesis. return false
5. After iterating through all the characters in the string

If the stack is empty return true (balanced parentheses). Otherwise return false.

PROGRAM:-

```
#include <stdio.h>
#include <stdbool.h>
char stack[30];
int top=0;
bool spec=0;
void push(char x){
    if (top==30)
        printf("Full \n");

    else{
        stack[top]=x;
        top++;
    }
}
void pop(char x) {
    if( top==0)
        spec=1;

    else{
        if (x==stack[top-1]);
        top--;
    }
```

```
int main()
{
char temp[30];
  scanf("%s", temp);
for(int i=0; temp[i]!='\0'; i++)

{
if(temp[i]=='(' || temp[i]=='{' || temp[i]=='[')
    push(temp[i]);
else if(temp[i]==')' || temp[i]=='}' || temp[i]==']')
        pop(temp[i]);

  }
for(int i=0; i<top; i++) {
        printf("%c", stack[i]);

}
if(top!=0)
  printf("unbalanced");
else
  printf("balanced");
  return 0;

}
```

Output:-

{
{ unbalanced.

Result:- Thus the program is successfully implemented for balancing parathesis using stack.

28/4/23 Q. Linked list implementation of queue.

Aim:-
C program for linked list implementation of queue.

ALGORITHM:-

1. Create a node with the given value and set the node's pointer is null
2. Check whether queue is empty.
3. If it is empty, set front and rear to new node.
4. Else, set the pointer of rear to new node and make rear as the new node.

PROGRAM:-

```
#include<stdio.h>
struct node{
  int data;
  struct node* next;
  struct node *front=NULL;
  struct node *rear=NULL;
void enqueue(struct node *ptr, int val){
  ptr=(struct node *)malloc(sizeof(struct node));
  ptr->data=value;
  ptr->next=null;
  if(front==NULL && rear==null){
    front=rear=ptr;
}
```

```
else
{
 rear->next=ptr;
  rear=ptr;
 }
printf("node is inserted: %d\n", value);

}
int dequeue()

{
 if (front==NULL)

 }
}
int main()

 {
  int ch, value;
  printf("\n 1n enqueue \n2 dequeue \n3 display \n4 EXIT
  switch (choice)

 {

case 1:-
    printf("Enter the value to insert");
    scanf("%d", &value);
    enqueue(value);
    break;
```

```
case 2:-
printf("removed element :%d\n");
break;

case 3:
display();
break;
case 4:
exit(0);

break;
default:
printf("wrong choice\n");

}
}
return 0;

}
```

OUTPUT:-

Enter the choice:1

Enter no: 123

Enter your choice=1

Enter no:65

Enter your choice:- 3

123
65

RESULT:-

Thus the program successfully implemented queue linked list implementation of queue.

28/10/23 10. Polynomial addition

AIM:-

Write a program for polynomial addition using linked list.

Algorithm:-

- Create a new linked list, numbered to store the resultant list.
- Traverse both lists until one of them is null.
- If any list is null insert that remaining node of another list in resultant list.
- Otherwise compare to degree of nodes
- If both are equal insert in resultant
- If equal one grades than another one then insert in resultant.

PROGRAM:-

```
#include <stdio.h>
#include <stdlib.h>
struct node {
        int coef;
        int exp;
        struct node *next;
};
```

```
typedef struct node node;
void insert(node **poly, int coef, int exp){
    node *temp = (node *)malloc(sizeof(node));
{
void print(node *poly){
    if(poly == NULL){
        printf("0\n");
        return;
    }
node *current = poly;
    while(current != NULL){
        printf("%d x^%d", current->coef, current->exp);
        if(current->next != NULL){
            printf("+");
        }
        current = current->next;
    }
    printf("\n");
}
node* add(node* poly1, node* poly2){
    node* result = NULL;
}
int main(){
```

```
node* poly1=NULL;
    insert(&poly1,3,4);
    insert(&poly2,3,2);
    insert(&poly1,1,0);
    printf("First polynomial:");
    print(poly1);
    printf("second polynomial:");
    print(poly);
    node* result=add(poly1,poly2)
    printf("Result:");
    print(result);
    return 0;
}
```

OUTPUT:-

First polynomial: $3x^4 + 3x^2 + 1x^0$.

Second polynomial: $4x^4 + 2x^2 + 1x^1$

Result: $7x^4 + 5x^2 + 1x^1 + 1x^0$.

RESULT:-

Thus the program is successfully implemented for addition of polynomials using linked list.