

**DEBRE MARKOS UNIVERSTITY**

**Institute of Technology**

**School of computing**

**Data Structure and Algorithm**

|  |  |  |
| --- | --- | --- |
| S.NO | NAME | ID NO |
| 1 | Yeabsira Belay | 1306882 |
| 2 | Netsanet Melese | 1403427 |
| 3 | Habtamu Alelign | 1306790 |
| 4 | Natnael Degu | 1307367 |
| 5 | Baweke Mekonen | 1305095 |

Date 26/07/2015E.C.

Submitted to: Mr. Ayalneh B.

1, palindrome cheker

#include <iostream>

#include <stack>

using namespace std;

struct node {

char word;

node \*link;

};

node \*top = NULL;

void push(char s){

node \*temp = new node;

temp -> word = s;

temp -> link = NULL;

if(top == NULL){

top = temp;

}

else {

node \*temp1 = top;

top = temp;

temp -> link = temp1;

}

}

void pop() {

if(top == NULL ) {

cout<<"the stack is empty";

return;

}

else {

node \*del = top;

top = top -> link;

delete del;

}

}

bool isPalindrome(string str){

for (int i = 0; i < str.size(); i++)

// Pushing all characters onto the stack

push(str[i]);

// Comparing characters from both ends of the string

for (int i = 0; i < str.size(); i++) {

if (top -> word != str[i])

return false;

pop();

}

return true;

}

int main() {

string str;

int option = 1;

while(option != 0){

cout << "Enter a string: ";

cin >> str;

if (isPalindrome(str))

cout << str << " is a palindrome\n";

else

cout << str << " is not a palindrome\n";

cout<<"do you want to try again if so enter '1' if not enter '0' ";

cin>>option;

}

return 0;

}

2, Expression converter

#include <bits/stdc++.h>

using namespace std;

struct inverser {

char ch;

inverser \*link;

};

inverser \*top = NULL;

void push(char ch){

inverser \*temp = new inverser;

temp -> ch = ch;

temp -> link = NULL;

if(top == NULL){

top = temp;

}

else {

inverser \*temper = top;

top = temp;

temp -> link = temper;

}

}

void pop() {

if(top == NULL ) {

cout<<"the stack is empty";

}

else {

inverser \*del = top;

top = top -> link;

delete del;

}

}

int prec(char c) {

if (c == '^')

return 3;

else if (c == '\*' || c == '/')

return 2;

else if (c == '+' || c == '-')

return 1;

else

return -1; // For brackets

}

string infixToPostfix(string exp) {

string postfix = "";

//inverser \*temp1 = top;

for (int i = 0; i < exp.length(); i++) {

char c = exp[i];

// If scanned character is an operand, add to output string

if (isalnum(c))

postfix += c;

// If scanned character is '(' push it to stack

else if (c == '(')

push(c);

// If scanned character is ')' pop it from stack and append its contents to output

else if (c == ')') {

while (top != NULL && top -> ch != '(') {

postfix += top -> ch;

pop();

}

if (top != NULL)

pop();

}

// If scanned character is an operator, pop and append to output

// those operators in the stack of equal or higher precedence.

else {

while (top != NULL && prec(c) <= prec(top -> ch)) {

postfix += top -> ch;

pop();

}

push(c);

}

}

// Pop remaining operators from the stack and add to output

while (top != NULL) {

postfix += top -> ch;

pop();

}

return postfix;

}

int main() {

string exp;

char say;

menu:

cout << "Enter an infix expression: ";

cin >> exp;

string postfix = infixToPostfix(exp);

cout << "The corresponding postfix expression is: " << postfix << endl;

cout<<"do you want to try again ('y' / 'n')\n";

cin>>say;

if(say == 'y') {

goto menu;

}

else

return 0;

return 0;

}

3, linked list

#include <iostream>

using namespace std;

// Node structure

struct node {

int data;

node\* next;

};

node\* head = NULL;

// Function to create a new node

node\* getNewNode(int data) {

node\* newNode = new node();

newNode->data = data;

newNode->next = NULL;

newNode->next = head;

head = newNode;

return newNode;

}

// Function to insert a node at the beginning of the list

void insertAtBeginning(int data) {

node\* newNode = getNewNode(data);

// newNode->next = head;

// head = newNode;

cout << data << " inserted at beginning successfully.\n";

}

// Function to insert a node at the end of the list

void insertAtEnd(int data) {

node\* newNode = new node;

newNode -> data = data;

newNode -> next = NULL;

if(head == NULL){

head = newNode;

cout << data << " inserted at end successfully.\n";

}

else {

node \*temp = head;

while (temp -> next != NULL)

temp = temp -> next;

temp -> next = newNode;

cout << data << " inserted at end successfully.\n";

}

}

// Function to insert a node at any position

void insertAtPosition(int data, int position) {

node\* newNode = new node;

newNode -> data = data;

if (position < 1) {

cout << "Invalid position. Please enter a valid position.\n";

return;

}

else if (position == 1) {

newNode->next = head;

head = newNode;

cout << data << " inserted at position " << position << " successfully.\n";

return;

}

else {

node\* temp = head;

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

if (temp == NULL) {

cout << "Position out of bounds. Please enter a valid position.\n";

return;

}

temp = temp->next;

}

newNode->next = temp->next;

temp->next = newNode;

cout << data << " inserted at position " << position << " successfully.\n";

}

}

// Function to sort the list

void sortList() {

node\* current = head;

node\* index = NULL;

int temp;

if (head == NULL) {

return;

}

else {

while (current != NULL) {

index = current->next;

while (index != NULL) {

if (current->data > index->data) {

temp = current->data;

current->data = index->data;

index->data = temp;

}

index = index->next;

}

current = current->next;

}

}

cout << "List sorted successfully.\n";

}

// Function to delete a particular node

void deleteNode(int key) {

node\* temp = head;

node\* prev = NULL;

if (temp != NULL && temp->data == key) {

head = temp->next;

delete temp;

cout << key << " deleted successfully.\n";

return;

}

while (temp != NULL && temp->data != key) {

prev = temp;

temp = temp->next;

}

if (temp == NULL) {

cout << key << " not found.\n";

return;

}

prev->next = temp->next;

delete temp;

cout << key << " deleted successfully.\n";

}

// Function to search for an element in the list

void searchElement(int key) {

node\* temp = head;

while (temp != NULL) {

if (temp->data == key) {

cout << key << " found.\n";

return;

}

temp = temp->next;

}

cout << key << " not found.\n";

}

// Function to display the list

void display() {

if (head == NULL) {

cout << "List is empty.\n";

return;

}

node\* temp = head;

while (temp != NULL) {

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

temp = temp->next;

}

cout << endl;

}

// Function to reverse the list

void reverse() {

node\* current = head;

node \*prev = NULL, \*next = NULL;

while (current != NULL) {

next = current->next;

current->next = prev;

prev = current;

current = next;

}

head = prev;

cout << "List reversed successfully.\n";

}

int main() {

int choice, data, position, key;

while (true) {

cout <<endl;

cout << "1. Create node\n2. Insert node at beginning\n3. Insert node at end\n4. Insert node at any position\n5. Sort list\n6. Delete a particular node\n7. Search element from list\n8. Display list\n9. Reverse list\n0. Exit\n";

cout << "Please enter your choice: ";

cin >> choice;

switch (choice) {

case 1:

cout << "Enter data: ";

cin >> data;

getNewNode(data);

break;

case 2:

cout << "Enter data: ";

cin >> data;

insertAtBeginning(data);

break;

case 3:

cout << "Enter data: ";

cin >> data;

insertAtEnd(data);

break;

case 4:

cout << "Enter data: ";

cin >> data;

cout << "Enter position: ";

cin >> position;

insertAtPosition(data, position);

break;

case 5:

sortList();

break;

case 6:

cout << "Enter key to delete: ";

cin >> key;

deleteNode(key);

break;

case 7:

cout << "Enter element to search: ";

cin >> key;

searchElement(key);

break;

case 8:

display();

break;

case 9:

reverse();

break;

case 0:

exit(0);

default:

cout << "Invalid choice. Please enter a valid choice.\n";

break;

}

}

return 0;

}

4, stack

#include <iostream>

using namespace std;

const int MAX\_SIZE = 100;

int top = -1;

int stack[MAX\_SIZE];

// Function to push an element onto the stack

void push(int element) {

if (top == MAX\_SIZE - 1) {

cout << "Stack overflow! Cannot push element " << element << " onto the stack.\n";

}

else {

top++;

stack[top] = element;

cout << "Element " << element << " pushed onto the stack successfully.\n";

}

}

// Function to pop an element from the stack

void pop() {

if (top == -1) {

cout << "Stack underflow! Cannot pop element from the stack.\n";

}

else {

int element = stack[top];

top--;

cout << "Element " << element << " popped from the stack successfully.\n";

}

}

// Function to display the contents of the stack

void display() {

if (top == -1) {

cout << "Stack is empty.\n";

}

else {

cout << "Contents of the stack are: ";

for (int i = top; i >= 0; i--) {

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

}

cout << endl;

}

}

int main() {

int choice, element;

while (true) {

cout << "\n";

cout << "1. enter '1' Push elements onto the stack\n2. Pop elements from the stack\n3. Display the contents of the stack\n4. Exit\n";

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1:

cout << "Enter element to push onto the stack: ";

cin >> element;

push(element);

break;

case 2:

pop();

break;

case 3:

display();

break;

case 4:

exit(0);

default:

cout << "Invalid choice. Please enter a valid choice.\n";

break;

}

}

return 0;

}

5, priority queue

int main() {

int choice;

do {

// Displaying menu of operations on queue

cout << endl;

cout << "--------------------------------\n";

cout << "1. To insert element to queue\n";

cout << "2. To delete element from queue\n";

cout << "3. To display all the elements of queue\n";

cout << "4. To exit\n";

cout << "\nEnter your choice: ";

cin >> choice;

switch(choice) {

case 1:

insertToPriorityQueue();

break;

case 2:

deleteFromPriorityQueue();

break;

case 3:

displayprQueue();

break;

case 4:

break;

default:

cout << "\nInvalid choice!\n";

}

} while (choice != 4);

return 0;

}

6, double ended queue

#include <iostream>

using namespace std;

// deque to store elements of the queue

struct node {

int data;

node \*prev, \*next;

};

node \*front = NULL, \*rear = NULL;

// Function to insert an element at the front of the queue

void enqueueFront() {

int element;

cout << "Enter element to insert at front of queue: ";

cin >> element;

node \*temp = new node;

temp -> data = element;

temp -> next = temp -> prev = NULL;

if(front == NULL){

front = temp;

rear = temp;

}

else {

node \*temp1 = front;

front = temp;

front -> next = temp1;

temp1 -> prev = temp;

}

cout<< "Element " << element << " added to front of queue successfully.\n";

}

// Function to insert an element at the back of the queue

void enqueueBack() {

int element;

cout << "Enter element to insert at back of queue: ";

cin >> element;

node \*temp = new node;

temp -> data = element;

temp -> next = temp -> prev = NULL;

if(rear == NULL){

front = rear = temp;

}

else {

node \*temp1 = rear;

rear = temp;

rear -> prev = temp1;

temp1 -> next = temp;

}

cout << "Element " << element << " added to back of queue successfully.\n";

}

// Function to delete an element from the front of the queue

void dequeueFront() {

if (front == NULL) {

cout << "Queue is empty.\n";

}

else {

node \*temp = front;

front = front -> next;

int element = temp -> data;

delete temp;

cout << "Element " << element << " deleted from front of queue successfully.\n";

}

}

// Function to delete an element from the back of the queue

void dequeueBack() {

if (rear == NULL) {

cout << "Queue is empty.\n";

}

else {

node \*temp = rear;

rear = rear -> prev;

rear -> next = NULL;

int element = temp -> data;

delete temp;

cout << "Element " << element << " deleted from back of queue successfully.\n";

}

}

// Function to display all the elements of the queue

void displayQueue() {

if (front == NULL && rear == NULL)

{

cout<<"queue is empty";

}

else {

node \*temp = front;

cout<<"\t\t elements of the queue \n";

while (temp != NULL)

{

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

temp = temp -> next;

}

}

}

int main() {

int choice;

while (true) {

cout << "\n\nMenu Operations on queue\n";

cout << "1. Insert element at front of queue\n2. Insert element at back of queue\n3. Delete element from front of queue\n4. Delete element from back of queue\n5. Display all the elements of queue\n6. Exit\n";

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1:

enqueueFront();

break;

case 2:

enqueueBack();

break;

case 3:

dequeueFront();

break;

case 4:

dequeueBack();

break;

case 5:

displayQueue();

break;

case 6:

exit(0);

default:

cout << "Invalid choice. Please enter a valid choice.\n";

break;

}

}

return 0;

}