# TASK 1:

#include<iostream>

#include<string>

using namespace std;

struct Node

{

int value;

Node\* next;

Node()

{

next = NULL;

value = NULL;

}

Node(int x) :Node()

{

value = x;

}

};

class linkedList

{

Node\* head;

public:

linkedList()

{

head = NULL;

}

void ins()

{

Node\* temp = head;

Node\* newnode = new Node;

if (temp == NULL)

{

head = newnode;

return;

}

while (temp->next != NULL)

{

temp = temp->next;

}

temp->next = newnode;

}

bool Even(int x)

{

Node\* temp = head;

int count = 0;

if (x % 2 == 0)

{

while (temp != NULL)

{

if (count % 2 == 0)

{

if (temp->value == NULL)

{

temp->value = x;

return 1;

}

}

count++;

temp = temp->next;

}

}

return 0;

}

bool Odd(int x)

{

Node\* temp = head;

int count = 0;

if (x % 2 != 0)

{

while (temp != NULL)

{

if (count % 2 != 0)

{

if (temp->value == NULL)

{

temp->value = x;

return 1;

}

}

count++;

temp = temp->next;

}

}

return 0;

}

bool Insert(int x)

{

bool y;

int z = 0;

while (x)

{

if (x == 2)

{

cout << "All indexs are Full...... ";

return 0;

}

if (x % 2 == 0)

{

y = Even(x);

if (y == 0)

{

cout << "All Even Index Are Full Enter Odd Value : ";

cin >> x;

x++;

}

else

return 1;

}

else

{

y = Odd(x);

if (y == 0)

{

cout << "All Even Index Are Full Enter even Value : ";

cin >> x;

x++;

}

else

return 1;

}

if (x == 2)

{

cout << "All indexs All full...... ";

return 0;

}

}

}

void disp()

{

Node\* temp = head;

while (temp)

{

cout << temp->value << endl;

temp = temp->next;

}

cout << endl;

}

};

int main()

{

linkedList obj;

for (int i = 0; i < 10; i++)

{

obj.ins();

}

bool flag = true;

bool y;

int value;

while (flag)

{

cout << "Enter Value : ";

cin >> value;

y = obj.Insert(value);

if (y == 0)

flag = false;

}

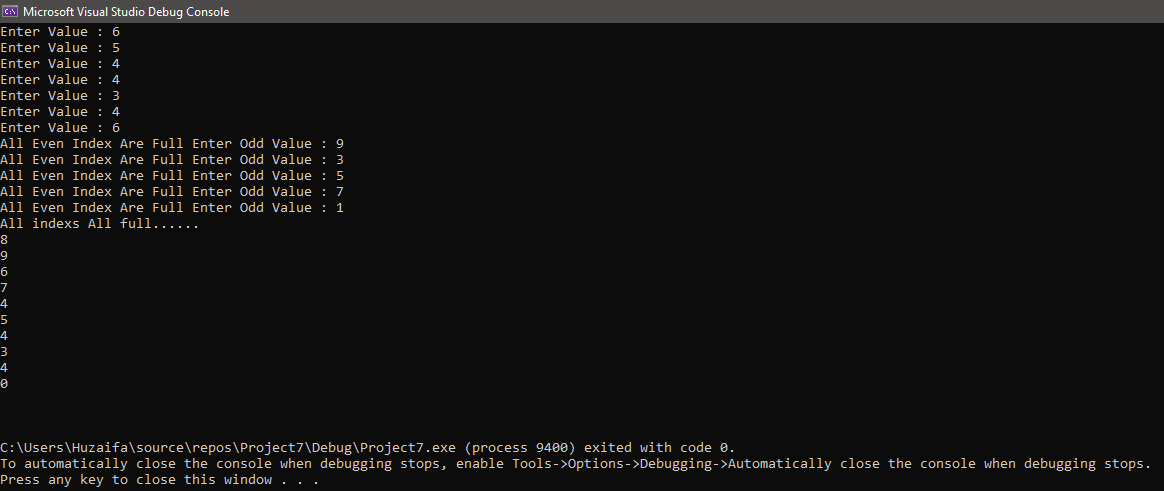
cout << endl;

obj.disp();

cout << endl;

}

# OUTPUT:



# TASK 2:

#include<iostream>

#include<string>

using namespace std;

struct Node

{

int value;

Node\* next;

Node()

{

next = NULL;

value = NULL;

}

Node(int x) :Node()

{

value = x;

}

};

class linkedList

{

Node\* head;

public:

linkedList()

{

head = NULL;

}

void Insert(int x)

{

Node\* newnode = new Node(x);

if (head == NULL)

{

head = newnode;

return;

}

Node\* temp = head;

while (temp->next != NULL)

{

temp = temp->next;

}

temp->next = newnode;

}

void ValueDiv()

{

Node\* temp = head;

if (temp == NULL)

{

return;

}

int count;

while (temp)

{

count = 1;

cout << temp->value << " : ";

while (count < temp->value)

{

if (temp->value % count == 0)

{

cout << count << " ";

}

count++;

}

cout << endl;

temp = temp->next;

}

}

};

int main()

{

linkedList obj;

int x;

bool flag = true;

char ch;

while (flag)

{

cout << "Enter value : ";

cin >> x;

obj.Insert(x);

cout << "Do you want to enter more [Y|N] : ";

cin >> ch;

if (ch == 'n' || ch == 'N')

flag = false;

}

obj.ValueDiv();

cout << endl;

}

# OUTPUT:

# TASK 3:

#include<iostream>

using namespace std;

struct Node

{

int value;

Node\* next;

Node()

{

next = NULL;

value = 0;

}

Node(int x) :Node()

{

value = x;

}

};

class linkedList

{

Node\* head;

public:

linkedList()

{

head = NULL;

}

void insertNodeAtEnd(int x)

{

Node\* newnode = new Node(x);

if (head == NULL)

{

head = newnode;

return;

}

else {

Node\* temp = head;

while (temp->next != NULL)

{

temp = temp->next;

}

temp->next = newnode;

return;

}

}

void insertNodeAtMiddle(int x, int key)

{

Node\* temp = head;

int y = 1;

while (temp && y != key && key > 2)

{

temp = temp->next;

y++;

}

if (temp)

{

Node\* newnode = new Node(x);

if (key == 1)

{

newnode->next = head;

head = newnode;

return;

}

if (key == 2)

{

newnode->next = head->next;

head->next = newnode;

return;

}

Node\* temp = head;

while (temp->next != temp)

{

temp = temp->next;

}

newnode->next = temp;

temp->next = newnode;

return;

}

cout << endl << "Key is out of bound..... " << endl;

return;

}

void InsertAtBeg(int x)

{

Node\* newnode = new Node(x);

if (head == NULL)

{

head = newnode;

return;

}

newnode->next = head;

head = newnode;

return;

}

bool deleFirstNode()

{

if (head == NULL)

return 0;

Node\* temp = head;

head = head->next;

delete temp;

return 1;

}

bool deleNode(int key)

{

Node\* temp = head;

int y = 1;

while (temp && y != key && key > 2)

{

temp = temp->next;

y++;

}

if (temp)

{

Node\* temp;

if (key == 1)

{

temp = head;

head = head->next;

delete temp;

return 1;

}

if (key == 2)

{

temp = head->next;

head->next = temp->next;

delete temp;

return 1;

}

Node\* temp1 = head;

while (temp1->next != temp)

{

temp1 = temp1->next;

}

temp1->next = temp->next;

delete temp;

return 1;

}

cout << endl << "Node does not exist in the list...." << endl;

return 0;

}

bool deleEnd()

{

if (head == NULL)

return 0;

Node\* temp = head;

while (temp->next != NULL)

{

temp = temp->next;

}

Node\* temp1 = head;

while (temp1->next != temp)

{

temp1 = temp1->next;

}

temp1->next = temp->next;

delete temp;

return 1;

}

void disp()

{

Node\* temp = head;

if (temp == NULL)

{

cout << "list is Empty....." << endl;

return;

}

while (temp)

{

cout << temp->value << endl;

temp = temp->next;

}

}

bool Search(int x)

{

Node\* temp = head;

while (temp && x != temp->value)

{

temp = temp->next;

}

if (temp)

return 1;

return 0;

}

linkedList\* Merge(linkedList\* obj, linkedList\* obj1)

{

Node\* temp = obj->head;

while (temp->next != NULL)

{

temp = temp->next;

}

temp->next = obj1->head;

return obj;

}

};

int main()

{

linkedList obj, obj1, obj2;

int x, key;

bool flag = true;

char C;

while (flag)

{

cout << "Enter value : ";

cin >> x;

obj.InsertAtBeg(x);

cout << "do you want to Enter more [Y|N] : ";

cin >> C;

if (C == 'n' || C == 'N')

flag = false;

}

cout << "Enter index : ";

cin >> key;

cout << "Enter value : ";

cin >> x;

obj.insertNodeAtMiddle(x, key);

flag = obj.deleEnd();

if (!flag)

{

cout << "List is empty.." << endl;

}

flag = obj.deleFirstNode();

if (!flag)

{

cout << "list is empty..." << endl;

}

cout << "Display...." << endl;

for (int i = 0; i < 10; i++)

{

obj1.InsertAtBeg(i);

obj2.InsertAtBeg(i + 2);

}

obj.disp();

cout << endl << "Merging Two Link list : " << endl;

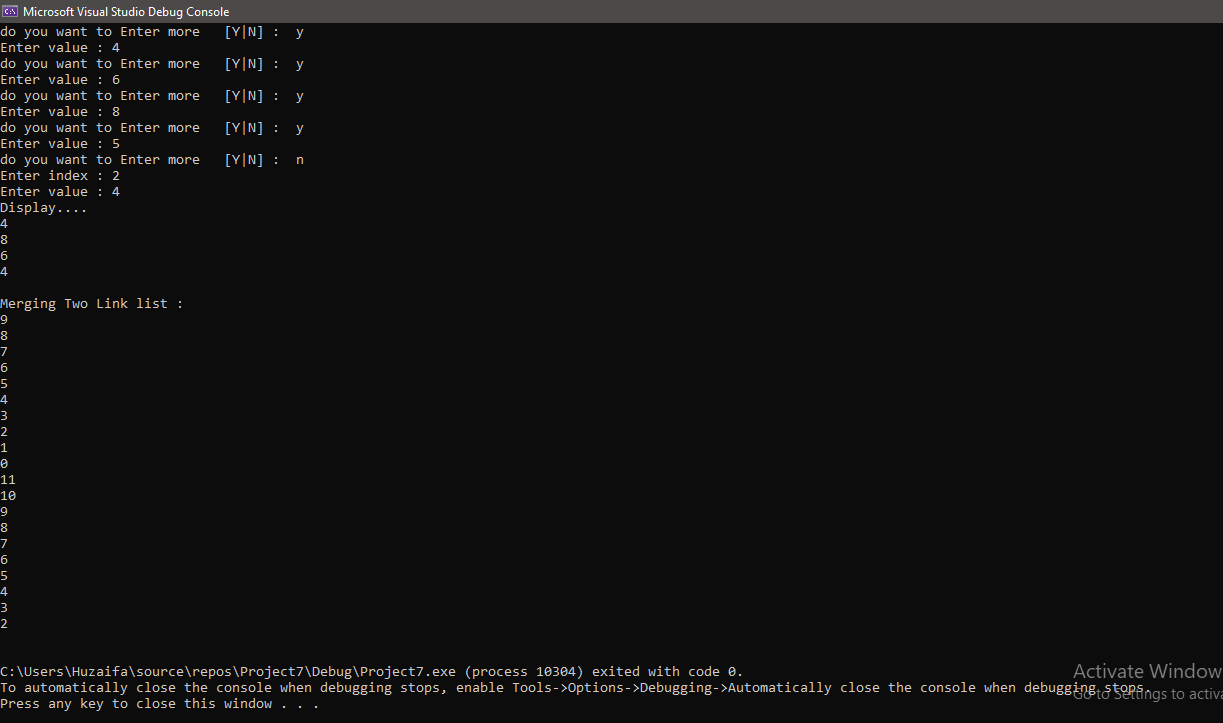
obj1.Merge(&obj1, &obj2);

obj1.disp();

cout << endl;

}

# OUTPUT:



# TASK 5:

#include<iostream>

using namespace std;

struct Node

{

int value;

Node\* next;

Node\* prev;

Node()

{

prev = next = NULL;

value = 0;

}

Node(int x) :Node()

{

value = x;

}

};

class linkedList

{

Node\* head;

Node\* tail;

public:

linkedList()

{

head = tail = NULL;

}

~linkedList()

{

Node\* ptr = head;

while (head)

{

head = head->next;

delete ptr;

ptr = head;

}

}

void insertNodeAtEnd(int x)

{

Node\* newnode = new Node(x);

if (head == NULL)

{

head = tail = newnode;

return;

}

else

{

tail->next = newnode;

newnode->prev = tail;

tail = newnode;

return;

}

}

void insertNodeAtMiddle(int x, int key)

{

Node\* temp = head;

int y = 1;

while (temp && y != key && key > 2)

{

temp = temp->next;

y++;

}

if (temp)

{

Node\* newnode = new Node(x);

if (key == 1)

{

newnode->next = head;

head->prev = newnode;

head = newnode;

return;

}

if (key == 2)

{

newnode->next = head->next;

head->next->prev = newnode;

head->next = newnode;

newnode->prev = head;

return;

}

Node\* temp1 = temp->prev;

newnode->next = temp;

temp->prev = newnode;

temp1->next = newnode;

newnode->prev = temp1;

return;

}

cout << endl << "Key is out of bound..... " << endl;

return;

}

void InsertAtBeg(int x)

{

Node\* newnode = new Node(x);

if (head == NULL)

{

head = tail = newnode;

return;

}

newnode->next = head;

head->prev = newnode;

head = newnode;

return;

}

bool deleFirstNode()

{

if (head == NULL)

return 0;

Node\* temp = head;

head = head->next;

delete temp;

head->prev = NULL;

return 1;

}

bool deleNode(int key)

{

Node\* curr = head;

int y = 1;

while (curr && y != key && key > 2)

{

curr = curr->next;

y++;

}

if (curr)

{

Node\* temp;

if (key == 1)

{

temp = head;

head = head->next;

delete temp;

return 1;

}

if (key == 2)

{

temp = head->next;

head->next = temp->next;

temp->next->prev = head;

delete temp;

return 1;

}

Node\* temp1 = curr->prev;

temp1->next = curr->next;

curr->next->prev = temp;

delete curr;

return 1;

}

cout << endl << "Node does not exist in the list...." << endl;

return 0;

}

bool deleEnd()

{

if (tail == NULL)

return 0;

Node\* temp = tail->prev;

delete tail;

tail = temp;

tail->next = NULL;

return 1;

}

void disp()

{

Node\* temp = head;

if (temp == NULL)

{

cout << "list is Empty....." << endl;

return;

}

while (temp)

{

cout << temp->value << endl;

temp = temp->next;

}

}

void dispRev()

{

Node\* temp = tail;

if (temp == NULL)

{

cout << "list is Empty....." << endl;

return;

}

while (temp)

{

cout << temp->value << endl;

temp = temp->prev;

}

}

bool Search(int x)

{

Node\* temp = head;

while (temp && x != temp->value)

{

temp = temp->next;

}

if (temp)

return 1;

return 0;

}

};

int main()

{

linkedList obj;

int x, key;

bool flag = true;

char C;

while (flag)

{

cout << "Enter value : ";

cin >> x;

obj.InsertAtBeg(x);

cout << "do you want to Enter more [Y|N] : ";

cin >> C;

if (C == 'n' || C == 'N')

flag = false;

}

cout << "Enter index : ";

cin >> key;

cout << "Enter value : ";

cin >> x;

obj.insertNodeAtMiddle(x, key);

flag = obj.deleEnd();

if (!flag)

cout << "List is empty.." << endl;

flag = obj.deleFirstNode();

if (!flag)

cout << "list is empty..." << endl;

cout << "Display...." << endl;

obj.disp();

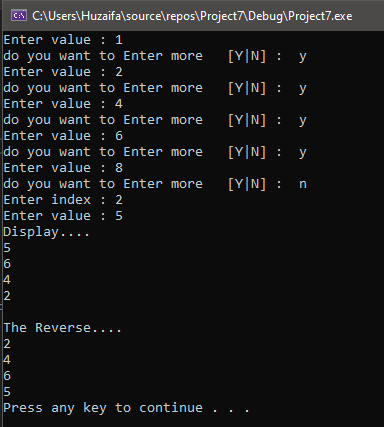
cout << endl << "The Reverse...." << endl;

obj.dispRev();

system("pause");

}

# OUTPUT:



# TASK 6:

#include<iostream>

#include<string>

using namespace std;

struct Node

{

char value;

Node\* next;

Node\* prev;

Node()

{

prev = next = NULL;

value = 0;

}

Node(char x) :Node()

{

value = x;

}

};

class linkedList

{

Node\* head;

public:

linkedList()

{

head = NULL;

}

void Insert(char x)

{

Node\* newnode = new Node(x);

if (head == NULL)

{

head = newnode;

newnode->next = head;

newnode->prev = head;

return;

}

Node\* temp = head->prev;

newnode->next = head;

temp->next = newnode;

newnode->prev = temp;

head->prev = newnode;

return;

}

bool check(char x)

{

switch (x)

{

case 'A':

case'a':

return 1;

case 'E':

case'e':

return 1;

case 'I':

case'i':

return 1;

case 'O':

case'o':

return 1;

case 'U':

case'u':

return 1;

case 'y':

case'Y':

return 1;

default:

return 0;

}

}

void Shift()

{

Node\* temp = head;

Insert('-');

if (check(temp->value))

{

Insert('w');

Insert('a');

Insert('y');

return;

}

temp = temp->next;

int x = 0;

while (!check(temp->value))

{

if (temp->next == head)

{

x = 1;

break;

}

temp = temp->next;

}

if (x == 1)

{

Insert('w');

Insert('a');

Insert('y');

return;

}

head = temp;

Insert('a');

Insert('y');

return;

}

void disp()

{

Node\* curr = head;

cout << curr->value;

curr = curr->next;

while (curr != head) {

cout << curr->value;

curr = curr->next;

}

cout << endl;

}

};

int main()

{

linkedList obj;

string st;

cout << "Enter a word : ";

cin >> st;

int x = 0;

while (st[x] != NULL)

{

obj.Insert(st[x]);

x++;

}

obj.Shift();

cout << "Pig latin form : ";

obj.disp();

cout << endl;

}

# OUTPUT:

# 

# TASK 7:

#include<iostream>

using namespace std;

#include <iostream>

using namespace std;

class node

{

public:

int data;

node\* next;

int n;

};

class LinkedList

{

node\* head;

node\* temp;

int n2;

public:

LinkedList()

{

head = NULL;

temp = NULL;

n2 = 0;

}

bool IsEmpty()

{

if (head == NULL)

{

return true;

}

else

{

return false;

}

}

void insert(int value)

{

if (IsEmpty())

{

head = new node;

head->data = value;

head->next = NULL;

head->n = ++n2;

return;

}

else

{

temp = head;

while (temp->next != NULL)

temp = temp->next;

}

temp->next = new node;

temp->next->data = value;

temp->next->n = ++n2;

temp->next->next = NULL;

}

bool insertAfterKey(int key, int value)

{

if (IsEmpty())

{

cout << "List is Empty" << endl;

return false;

}

else

{

node\* temp2;

temp = head;

while (temp->data != key)

{

temp = temp->next;

if (temp == NULL)

{

cout << "Key not found" << endl;

return false;

}

}

temp2 = new node;

temp2->data = value;

temp2->next = temp->next;

temp->next = temp2;

}

return true;

}

void display()

{

if (IsEmpty())

return;

else

{

temp = head;

while (temp != NULL)

{

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

temp = temp->next;

}

}

cout << endl;

}

void deleteKey(int key)

{

temp = head;

node\* temp2;

temp2 = temp;

if (head->next == NULL)

{

delete head;

head = NULL;

return;

}

do

{

temp2 = temp;

temp = temp->next;

} while (temp->data != key);

temp2->next = temp->next;

if (temp2 == temp2->next)

{

temp2->next = NULL;

delete temp;

return;

}

delete temp;

}

void process\_count()

{

node\* temp = head;

cout << "Process In Linked List : ";

do

{

cout << temp->n;

cout << " ";

temp = temp->next;

if (temp == NULL)

break;

} while (temp != head);

cout << "-";

}

void circular()

{

node\* temp = head;

while (temp->next != NULL)

{

temp = temp->next;

}

temp->next = head;

}

void process(int timeslice)

{

node\* temp = head;

int total\_time = 0;

circular();

while (temp != NULL)

{

if (temp->data <= timeslice)

{

total\_time = total\_time + temp->data;

process\_count();

cout << " Process #" << temp->n << " Executed For " << temp->data << " Seconds.";

cout << "\n\nProcess #" << temp->n << " Has Been Executed. Time Taken : " << total\_time << " Seconds.";

node\* ptr = temp;

if (temp->data == head->data)

{

temp = head->next;

deleteKey(ptr->data);

head = temp;

}

else

{

temp = temp->next;

deleteKey(ptr->data);

}

cout << endl << endl;

continue;

}

else

{

temp->data = temp->data - timeslice;

process\_count();

cout << " Process #" << temp->n << " Executed For " << timeslice << " Seconds.";

total\_time = total\_time + timeslice;

cout << endl << endl;

}

temp = temp->next;

}

}

};

int main()

{

int process, time, timeslice;

LinkedList obj;

cout << "Enter No Of Processes : ";

cin >> process;

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

{

cout << "Enter Process #" << i + 1 << " Execution Time : ";

cin >> time;

obj.insert(time);

}

cout << "Enter Time Slice : ";

cin >> timeslice;

cout << endl;

obj.process(timeslice);

# } OUTPUT: