**Table of Contents**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.No** | **Program Title** | **Page No.** | **Signature** |
| 1. | Write a C++ program to find whether the year is leap or not. | 1 |  |
| 2. | Write a C++ program to input sides of a triangle and check whether the triangle is valid or not? | 1 |  |
| 3. | Write a C++ program to print all prime numbers within a range using functions. | 1 |  |
| 4. | Write a C++ program to print sum of Fibonacci series using functions. | 2 |  |
| 5. | Write a C++ program to print the day on the date entered by the user | 2 |  |
| 6. | Write a C++ program of inheritance of same member function name in class A and B and call using A class object. | 3 |  |
| 7. | Write a C++ program of inheritance of same member function name in class A and B and call using A class object. | 3 |  |
| 8. | Write a C++ program to depict early binding in C++. | 3 |  |
| 9. | Write a C++ program to depict the use of delete in c++. | 4 |  |
| 10. | Write a C++ program to input value of Z. If z is 0, then float catch is called .If z is 1, then integer catch is called. If z is a 2, then character catch is called ,else default catch is called. | 4 |  |
| 11. | Write a C++ program to depict nested try and catch statement by checking the conditions for numerator and denominator. | 5 |  |
| 12. | Write a C++ program to create a vector and enter the values into the vector using push\_back() method. Input the values from the user until the choice is yes i.e y. | 5 |  |
| 13. | Write a C++ program to create a vector and enter the values into the vector by user. Print the first element of the vector using begin and iterator. | 6 |  |
| 14. | Write a C++ program to create a vector and enter the values into the vector by user. Then to print that vector, sort it and then reverse it. | 6 |  |
| 15. | Write a C++ program to create 2 lists by entering the values from the user. Print the 2 lists. Then sort the first list and reverse the 2nd list. Then merge these 2 lists and then sort that merged list and reverse it and finally print it. Find the maximum and minimum elements of the merged list and finally insert a number entered by the user into the list. | 7 |  |
| 16. | C++ program to demonstrate example of seekg() function. | 7 |  |
| 17. | C++ program to demonstrate example of tellp() function. | 8 |  |
| 18. | Write a C++ program to create a class named car with public members speed and milage, both of float type and private member brand .Public member functions are input() and output(). Input 3 different cars details and print the details in ascending order of their speed and milage separately. | 8 |  |
| 19. | Write a C++ program to create a class named square with private members as side and public member functions named input .Also create another class named rectangle with length and width and public member function input. The members of both the classes can be operated with the help of friend class. The compare function returns the value to the main function which is highest area among square and rectangle. | 9 |  |
| 20. | Write a C++ program to create class named complex private members real and imag. A parametrized constructor takes the value of real and imaginary part. ‘+’ operator is overloaded and sum of complex number is printed. | 10 |  |
| 21. | Write a C++ program to create class named coordinate with two members x and y declared as private .the value of x and y should be provided by parameterized constructor. Write a program to compute distance between two objects and you also need to find the greatest distance between them by overloading relational operators. | 10 |  |
| 22. | Write a C++ program to create a class university with public member functions input() and it derives class student. From class student, Classes sports and academics are derived and class subject also derives class academics. Finally class Result is derived from class sports and academics. All these classes have public member function named as input() which takes input according to the class and class result also has an output() member function to display the details. | 11 |  |
| 23. | Write a C++ program to create 2 lists by entering the values from the user. Print the 2 lists. Then sort the first list and reverse the 2nd list. Then merge these 2 lists and then sort that merged list and reverse it and finally print it. Find the maximum and minimum elements of the merged list and finally insert a number entered by the user into the list. | 12 |  |
| 24. | Write a C++ program for Linked List Deletion (Deleting a given key) | 13 |  |
| 25. | Write a C++ program for Linked List Deletion (Deleting a key at given position) | 14 |  |
| 26 | Write a C++ program to find Length of a Linked List | 15 |  |
| 27. | Write a C++ program to search an element in a Linked List | 15 |  |
| 28. | Write a C++ program to get Nth node in a Linked List | 16 |  |
| 29. | Write a C++ program that counts the number of times a given int occurs in a Linked List | 17 |  |
| 30. | Reverse a linked list. | 17 |  |
| 31. | Detect loop in a linked list | 18 |  |
| 32. | Merge two sorted Linked Lists | 19 |  |
| 33. | Function to check if a singly linked list is palindrome | 20 |  |
| 34. | Quick Sort with arrays | 20 |  |
| 35. | Merge Sort with arrays | 21 |  |
| 36. | Insertion Sort with arrays | 22 |  |
| 37. | Selection Sort with Arrays | 22 |  |
| 38. | Radix Sort with Arrays. | 23 |  |
| 39. | Bubble Sort with Arrays | 24 |  |
| 40. | Program to check whether the given expression is balanced or not. | 24 |  |
| 41. | Infix to postfix conversion using stacks | 25 |  |
| 42. | Infix to prefix conversion using stacks | 26 |  |
| 43. | Evaluation of prefix expression using stack | 27 |  |
| 44. | Evaluation of postfix expression using stack | 28 |  |
| 45. | Merge two sorted linked lists | 29 |  |
| 46. | Write a C++ program to print the middle of a given linked list | 30 |  |
| 47. | Implement queues using stack | 31 |  |
| 48. | Implement stacks using queues. | 32 |  |
| 49. | C++ program to create a file, write and read text in/from file. | 33 |  |
| 50. | Perform Push, Pop operations in stacks with linked list implementation | 33 |  |
| 51. | Insertion and deletion in linear queue | 34 |  |
| 52. | Insertion and deletion in circular queue | 36 |  |
| 53. | Perform Push, Pop operations in stacks with array implementation | 36 |  |
| 54. | Insertion in Circular linked list at a given position (begin, end, middle) | 38 |  |
| 55. | Insertion in Double linked list at a given position (begin, end, middle) | 38 |  |
| 56. | Remove duplicates from a sorted linked list. | 39 |  |
| 57. | Intersection point of two Linked Lists. | 40 |  |
| 58. | Remove duplicates from an unsorted linked list | 41 |  |
| 59. | Pairwise swap elements of a given linked list. | 41 |  |
| 60. | Move last element to front of a given Linked List | 42 |  |
| 61. | Segregate even and odd nodes in a Linked List. | 43 |  |
| 62. | Add two numbers represented by linked lists | 44 |  |
| 63. | Subtract two numbers represented by linked lists | 45 |  |
| 64. | Multiply two numbers represented by linked lists. | 46 |  |
| 65. | Intersection of two Sorted Linked Lists | 47 |  |
| 66. | Delete alternate nodes of a Linked List. | 48 |  |
| 67. | Quick Sort with double linked list. | 48 |  |
| 68. | Merge Sort with linked List. | 50 |  |
| 69. | Insertion Sort with linked List. | 50 |  |
| 70. | Bubble Sort with Linked List. | 51 |  |

**Program 1: Write a C++ program to find whether the year is leap or not.**

#include <iostream>

using namespace std;

bool checkYear(int year)

{

if (year % 400 == 0)

return true;

if (year % 100 == 0)

return false;

if (year % 4 == 0)

return true;

return false;

}

int main()

{

cout<<"Enter the year->";

int year;

cin>>year;

checkYear(year)?cout<<"LEAP YEAR":cout<<"Not Leap Year";

return 0;

}

**Program 2: Write a C++ program to input sides of a triangle and check whether the triangle is valid or not.**

#include<iostream>

using namespace std;

bool check(int a, int b, int c)

{

if (a + b <= c || a + c <= b || b + c <= a)

return false;

else

return true;

}

int main()

{

int a, b, c;

cin>>a>>b>>c;

if (check(a, b, c))

cout << "Valid";

else

cout << "Invalid";

}

**Program 3: Write a C++ program to print all prime numbers within a range using functions.**

#include<iostream>

using namespace std;

int main()

{

int low, high, i, flag;

printf("Enter two numbers(intervals): ");

cin>>low>>high;

cout<<"Prime numbers between are: "<<low<<" "<<high<<endl;

while (low < high)

{

flag = 0;

for(i = 2; i <= low/2; ++i)

{

if(low % i == 0)

{

flag = 1;

break;

}

}

if (flag == 0)

cout<<low<<" ";

++low;

}

return 0;

**Program 4: Write a C++ program to print sum of Fibonacci series using functions.**

#include<iostream>

using namespace std;

int calculateSum(int n)

{

if (n <= 0)

return 0;

int fibo[n+1];

fibo[0] = 0, fibo[1] = 1;

int sum = fibo[0] + fibo[1];

for (int i=2; i<=n; i++)

{

fibo[i] = fibo[i-1]+fibo[i-2];

sum += fibo[i];

}

return sum;

}

int main()

{

int n;

cin>>n;

cout << "Sum of Fibonacci numbers is : " << calculateSum(n) << endl;

return 0;

}

**Program 5: Write a C++ program to print the day on the date entered by the user.**

#include<iostream>

#include<string>

using namespace std;

int dayofweek(int d, int m, int y)

{

static int t[] = { 0, 3, 2, 5, 0, 3, 5, 1, 4, 6, 2, 4 };

y -= m < 3;

return ( y + y/4 - y/100 + y/400 + t[m-1] + d) % 7;

}

int main()

{

string re[7]={"Monday","Tuesday","Wednesday","Thrusday","Friday","Saturday","Sunday"};

int date,month,year;

cin>>date>>month>>year;

int day = dayofweek(date, month,year);

cout<<re[day-1];

return 0;

}

**Program 6: Write a C++ program of inheritance of same member function name in class A and B and call using A class object.**

#include<iostream>

using namespace std;

class A

{

public:

int foo(int c)

{

c = c+1;

return c;

}

};

class B : public A

{

public:

int foo(int c)

{

c = c-1;

return c;

}

};

int main()

{

A test;

cout<<test.foo(20);

}

**Program 7: Write a C++ program of inheritance of same member function name in class A and B and call using A class object.**

#include<iostream>

using namespace std;

class A

{

public:

int foo(int c)

{

c = c+1;

return c;

}

};

class B : public A

{

public:

int foo(int c)

{

c = c-1;

return c;

}

};

int main()

{

B test;

cout<<test.foo(20);

}

**Program 8: Write a C++ program to depict early binding in C++.**

#include<iostream>

using namespace std;

class A

{

int i;

public:

void out1()

{

cout<<"New";

}

void out2()

{

cout<<"Better";

}

};

class B:public A

{

int k;

private:

void out2()

{

cout<<"Welcome";

}

};

int main()

{

cout<"------"<<endl;

A \*a;

B b;

a=&b;

a->out1(),a->out2()

}

**Program 9: Write a C++ program to depict the use of delete in c++.**

#include<iostream>

using namespace std;

int main()

{

int x;

x=90;

int \*ptr1 = &x;

int \*ptr2 = new int;

int \*ptr3 = new int;

int \*ptr4 = NULL;

delete ptr1;

delete ptr2;

delete ptr3;

delete ptr4;

getchar();

return 0;

}

**Program 10 : Write a C++ program to input value of Z. If z is 0, then float catch is called .If z is 1, then integer catch is called. If z is a 2, then character catch is called ,else default catch is called.**

#include<iostream>

using namespace std;

int main()

{

int z;

cout<<"Enter the value of z->"<<endl;

cin>>z;

try

{

if(z==0)

{

throw 5.6;

}

if(z==1)

{

throw 5;

}

if(z==2)

{

throw 'a';

}

}

catch(float y)

{

cout<<"Number is float";

}

catch(int y)

{

cout<<"Number is Integer";

}

catch(char a)

{

cout<<"Character";

}

catch(...)

{

cout<<"Default Catch";}}

**Program 11 : Write a C++ program to depict nested try and catch statement by checking the conditions for numerator and denominator.**

#include<iostream>

using namespace std;

int main()

{

cout<<" "<<endl;

cout<<" "<<endl;

float num,deno;

cin>>num>>deno;

try

{

if(deno==0)

{

throw deno;

}

else

{

cout<<num/deno;

}

}

catch(float y)

{

cerr<<"Wrong Denominator";

}

catch(const char \* a)

{

cerr<<"V";

**Program 12: Write a C++ program to create a vector and enter the values into the vector using push\_back() method. Input the values from the user until the choice is yes i.e y.**

#include<iostream>

#include<vector>

#include<cstring>

using namespace std;

int main()

{

string ch;

int n,i;

vector <int> v;

do

{

cout<<"Enter the number->"<<endl;

cin>>n;

v.push\_back(n);

cout<<"Enter the y/n"<<endl;

cin>>ch;

}

while(ch =="y");

for(i=0;i<v.size();i++)

{

cout<<v[i];

}

}

**Program 13: Write a C++ program to create a vector and enter the values into the vector by user. Print the first element of the vector using begin and iterator.**

#include<iostream>

#include<vector>

using namespace std;

int main()

{

vector<int> v;

int p,i,n;

cout<<"Enter the number of the elements"<<endl;

cin>>p;

for(i=0;i<p;i++)

{

cout<<"Enter the number"<<endl;

cin>>n;

v.push\_back(n);

}

vector<int>::iterator z=v.begin();

cout<<\*z;

}

**Ouput:**

**Program 14: Write a C++ program to create a vector and enter the values into the vector by user. Then to print that vector, sort it and then reverse it.**

#include<algorithm>

#include<iostream>

#include<vector>

using namespace std;

int main()

{

vector<int> v;

int p,i,n;

cin>>p;

for(i=0;i<p;i++)

{

cin>>n;

v.push\_back(n);

}

vector<int>::iterator z=v.begin();

sort(v.begin(),v.end());

for(z=v.begin();z!=v.end();z++)

{

cout<<\*z;

}

vector<int>::iterator t;

reverse(v.begin(),v.end());

for(t=v.begin();t!=v.end();t++)

{

cout<<\*t;

}

}

**Ouput:**

**Program 15 : Write a C++ program to create 2 lists by entering the values from the user. Print the 2 lists. Then sort the first list and reverse the 2nd list. Then merge these 2 lists and then sort that merged list and reverse it and finally print it. Find the maximum and minimum elements of the merged list and finally insert a number entered by the user into the list.**

#include<list>

#include<iostream>

#include<algorithm>

using namespace std;

int main()

{

cout<<" "<<endl;

cout<<" "<<endl;

int n1,i,p1,p2,n2;

list<int> l1;

list<int> l2;

cout<<"Enter the element in the first list->";

cin>>n1;

for(i=0;i<n1;i++)

{

cin>>p1;

l1.push\_back(p1);

}

cout<<"Enter the element in the second list->";

cin>>n2;

for(i=0;i<n2;i++)

{

cin>>p2;

l2.push\_back(p2);

}

list<int>::iterator z1=l1.begin();

list<int>::iterator z2=l2.begin();

l1.sort(),l2.reverse(),l1.merge(l2),l1.sort();

for(z1=l1.begin();z1!=l1.end();z1++)

{

cout<<\*z1<<" "<<endl;

}

l1.reverse();

for(z1=l1.begin();z1!=l1.end();z1++)

{

cout<<\*z1<<" ";

}

list<int>::iterator z3;

z3=min\_element(l1.begin(),l1.end());

for(z3=l1.begin();z3!=l1.end();z3++)

{cout<<\*z3<<" ";}

**Program 16 : C++ program to demonstrate example of seekg() function.**

#include <fstream>

#include <iostream>

using namespace std;

int main (int argc, char\*\* argv)

{

fstream myFile("test.txt", ios::in | ios::out | ios::trunc);

myFile << "Hello World";

myFile.seekg(6, ios::beg);

char A[6];

myFile.read(A, 5);

A[5] = 0;

cout << "DONE" << endl;

myFile.close();}

**Output:**

**Program 17 : C++ program to demonstrate example of tellp() function.**

#include <iostream>

#include <fstream>

using namespace std;

int main()

{

fstream file;

file.open("myfile.txt", ios::out);

file << "geeksforgeeks";

cout << "the current position of pointer is :"

<< file.tellp() << endl;

file.close();

}

**Program 18 : Write a C++ program to create a class named car with public members speed and milage, both of float type and private member brand .Public member functions are input() and output(). Input 3 different cars details and print the details in ascending order of their speed and milage separately.**

#include<iostream>

using namespace std;

class car

{

public:

char brand[100];

public:

float speed;

float mileage;

public:

void input()

{

cin>>brand>>speed>>mileage;

}

void display()

{

cout<<"---------"<<endl;

cout<<brand<<endl;

cout<<speed<<endl;

cout<<mileage<<endl;

}

};

int main()

{

car c[10];

car temp;

int i,j;

for(i=0;i<3;i++)

{

c[i].input();

}

for(i=0;i<3;i++)

{

for(j=0;j<3-i-1;j++)

{

if(c[j].speed>c[j+1].speed)

{

temp=c[j];

c[j]=c[j+1];

c[j+1]=temp;

}

}

}

for(i=0;i<3;i++)

{

c[i].display();

}

}

**Output:**

**Program 19. Write a C++ program to create a class named square with private members as side and public member functions named input .Also create another class named rectangle with length and width and public member function input. The members of both the classes can be operated with the help of friend class. The compare function returns the value to the main function which is highest area among square and rectangle.**

#include<iostream>

using namespace std;

class rect;

class square

{

private:

int side;

public:

void input(){

cout<<"Enter the side of the square";

cin>>side;

}

friend int comp(square,rect);

};

class rect

{

public:

int len;

int widt;

public:

void input()

{

cout<<"Enter the length of the rect";

cin>>len;

cout<<"Enter the width of the rect";

cin>>widt;

}

};

int comp(square s1,rect r1){

int sarea,srect;

sarea=s1.side\*s1.side;

srect=r1.len\*r1.widt;

if(sarea>srect)

{

cout<<"Area of square is greater than rectangle"<<endl;

cout<<sarea;

}

else

{

cout<<"Area of rectangle is greater than square"<<endl;

cout<<srect;

}

}

int main(){

square s1;

rect r1;

s1.input();

r1.input();

comp(s1,r1);

}

**Program 20: Write a C++ program to create class named complex private members real and imag. A parametrized constructor takes the value of real and imaginary part. ‘+’ operator is overloaded and sum of complex number is printed.**

#include <iostream>

using namespace std;

class Complex

{

private:

float real;

float imag;

public:

Complex(): real(0), imag(0){ }

void input()

{

cout << "Enter real and imaginary parts respectively: ";

cin >> real;

cin >> imag;

}

Complex operator + (Complex c2)

{

Complex temp;

temp.real = real + c2.real;

temp.imag = imag + c2.imag;

return temp;

}

void output()

{

cout << "Output Complex number: " << real << "+" << imag << "i";

}

};

int main()

{

Complex c1, c2, result;

cout<<"Enter first complex number:\n";

c1.input();

cout<<"Enter second complex number:\n";

c2.input();

result = c1 + c2;

result.output();

return 0;

}

**Program 21 : Write a C++ program to create class named coordinate with two members x and y declared as private .the value of x and y should be provided by parameterized constructor. Write a program to compute distance between two objects and you also need to find the greatest distance between them by overloading relational operators.**

#include<iostream>

#include<cmath>

using namespace std;

class dist

{

private:

int x;

int y;

public:

dist()

{

}

void input()

{

cout<<"Enter the coordinates";

cin>>x>>y;

}

dist operator -(dist const &obj)

{

dist temp;

temp.x=(x-obj.x);

temp.y=(y-obj.y);

return(temp);

}

void output()

{

dist f;

f.x=(x\*x+y\*y);

cout<<"First Distance->";

cout<<sqrt(f.x)<<endl;

}

void output1()

{

dist g;

g.x=(x\*x+y\*y);

cout<<"Second Distance->";

cout<<sqrt(g.x)<<endl;

}

bool operator <(dist const &obj1)

{

if(x<obj1.x){

return true;

}else{

return false;

}

}

};

int main()

{

dist d1,d2,d,e,d3,d4;

d1.input(),d2.input(),d3.input(),d4.input();

d=d1-d2;

e=d3-d4;

d.output();

e.output1();

int z=d<e;

cout<<z;

**Program 22 : Write a C++ program to create a class university with public member functions input() and it derives class student. From class student, Classes sports and academics are derived and class subject also derives class academics. Finally class Result is derived from class sports and academics. All these classes have public member function named as input() which takes input according to the class and class result also has an output() member function to display the details.**

#include<iostream>

#include<cstring>

using namespace std;

class university

{

protected:

string uname;

public:

void input()

{

cout<<"Enter the University Name->"<<endl;

cin>>uname;

}

};

class student:virtual public university

{

protected:

int roll\_no;

string name;

public:

void input()

{

cout<<"Enter the roll\_no->"<<endl;

cin>>roll\_no;

cout<<"Enter the name of the student"<<endl;

cin>>name;

}

};

class subject

{

protected:

string s1,s2,s3;

public:

void input()

{

cout<<"Enter the name of the three subjects->"

cin>>s1>>s2>>s3;

}

};

class sports:virtual public student

{

protected:

int smarks;

public:

void input()

{

cout<<"Enter the sports Marks->"

cin>>smarks;

}

};

class academics:virtual public student,virtual public subject

{

protected:

int marks1,marks2,marks3;

public:

void input()

{

cout<<"Enter the marks of the three subjects";

cin>>marks1>>marks2>>marks3;

}

};

class result:virtual public sports,virtual public academics

{

protected:

int batch;

public:

void input()

{

cout<<"Enter the batch year"<<endl;

cin>>batch;

}

void output()

{

cout<<"University Name->"<<uname<<endl;

cout<<"Roll\_no->"<<roll\_no<<endl;

cout<<"Name->"<<name<<endl;

cout<<"Total>"<<smarks+marks1+marks2+marks3<<endl;

}

};

int main()

{

cout<<"---"<<endl;

result obj1;

obj1.university::input();

obj1.student::input();

obj1.sports::input();

obj1.academics::input();

obj1.input();

obj1.output();

}

**Output:**

**Program 23 : Write a C++ program to create 2 lists by entering the values from the user. Print the 2 lists. Then sort the first list and reverse the 2nd list. Then merge these 2 lists and then sort that merged list and reverse it and finally print it. Find the maximum and minimum elements of the merged list and finally insert a number entered by the user into the list.**

#include<list>

#include<iostream>

#include<algorithm>

using namespace std;

int main()

{

cout<<" "<<endl;

cout<<" "<<endl;

int n1,i,p1,p2,n2;

list<int> l1;

list<int> l2;

cout<<"Enter the element in the first list->";

cin>>n1;

for(i=0;i<n1;i++)

{

cin>>p1;

l1.push\_back(p1);

}

cout<<"Enter the element in the second list->";

cin>>n2;

for(i=0;i<n2;i++)

{

cin>>p2;

l2.push\_back(p2);

}

list<int>::iterator z1=l1.begin();

list<int>::iterator z2=l2.begin();

l1.sort(),l2.reverse(),l1.merge(l2),l1.sort();

for(z1=l1.begin();z1!=l1.end();z1++)

{

cout<<\*z1<<" "<<endl;

}

l1.reverse();

for(z1=l1.begin();z1!=l1.end();z1++)

{

cout<<\*z1<<" ";

}

list<int>::iterator z3;

z3=min\_element(l1.begin(),l1.end());

for(z3=l1.begin();z3!=l1.end();z3++)

{

cout<<\*z3<<" ";}

**Output:**

**Program 24 : Write a C++ program for Linked List Deletion (Deleting a given key)**

#include<iostream>

using namespace std;

struct node\* create();

struct node\* remove();

void travel();

struct node

{

int data;

struct node \*next;

}\*head;

int main()

{

cout<<"SEC->112"<<endl;

create();

remove();

travel();

}

struct node\* create()

{

int i;

char ch;

struct node \*newnode,\*temp;

head=new node;

cin>>head->data;

head->next=NULL;

temp=head;

cin>>ch;

while(ch=='y')

{

newnode=new node;

cin>>newnode->data;

newnode->next=NULL;

temp->next=newnode;

temp=temp->next;

cin>>ch;

}

return temp;

}

struct node\* remove()

{

int n;

cin>>n;

struct node \*ptr1;

struct node \*ptr2;

int d;

ptr1=head;

if(ptr1->data==n)

{

ptr1=head;

head=head->next;

delete ptr1;

return head;

}

while(ptr1->next->data!=n)

{

ptr1=ptr1->next;

}

ptr2=ptr1->next;

ptr1->next=ptr1->next->next;

delete ptr2;

return head;

}

void travel()

{

struct node \*temp;

temp=head;

while(temp!=NULL)

{

cout<<temp->data;

temp=temp->next;

}

}

**Output:**

**Program 25 :Write a C++ program for Linked List Deletion (Deleting a key at given position)**

#include<iostream>

using namespace std;

struct node\* create();

struct node\* remove();

void travel();

struct node

{

int data;

struct node \*next;

}\*head;

int main()

{

cout<<"SEC->112"<<endl;

create();

remove();

travel();

}

struct node\* create()

{

int i;

char ch;

struct node \*newnode,\*temp;

head=new node;

cin>>head->data;

head->next=NULL;

temp=head;

cin>>ch;

while(ch=='y')

{

newnode=new node;

cin>>newnode->data;

newnode->next=NULL;

temp->next=newnode;

temp=temp->next;

cin>>ch;

}

return temp;

}

struct node\* remove()

{

int p;

cout<<"Enter the position->"<<endl;

cin>>p;

struct node\* ptr1=head;

if(p==1)

{

head=head->next;

return head;

}

while(p>1)

{

ptr1=ptr1->next;

p--;

}

struct node\* ptr2=ptr1->next;

ptr1->next=ptr1->next->next;

delete(ptr2);

return head;

}

void travel()

{

struct node \*temp;

temp=head;

while(temp!=NULL)

{

cout<<temp->data;

temp=temp->next;

}

}

**Program 26 : Write a C++ program to find Length of a Linked List**

#include<math.h>

#include<iostream>

using namespace std;

struct node\* create(struct node\* start);

void check(struct node\* start);

struct node\* create(struct node\* start);

void print(struct node\* start);

void count(struct node\* start);

struct node

{

int data;

struct node \*next;

};

int main()

{

struct node\* start;

start=new node;

start->next=NULL;

create(start);

count(start);

}

struct node\* create(struct node\* start)

{

struct node \*newnode,\*temp;

cin>>start->data;

temp=start;

char ch;

cin>>ch;

while(ch=='y')

{

newnode=new node;

cin>>newnode->data;

newnode->next=NULL;

temp->next=newnode;

temp=newnode;

cin>>ch;

}

return start;

}

void count(struct node\* start)

{

struct node\* ptr=start;

int c=0;

while(ptr!=NULL)

{

c=c+1;

ptr=ptr->next;

}

cout<<"Length of linked list-> "<<c;

}

**Output:**

**Program 27 .Write a C++ program to search an element in a Linked List**

#include<math.h>

#include<iostream>

using namespace std;

struct node\* create(struct node\* start);

void search(struct node\* start);

void print(struct node\* start);

struct node

{

int data;

struct node \*next;

};

int main()

{

struct node\* start;

start=new node;

start->next=NULL;

create(start);

search(start);

}

struct node\* create(struct node\* start)

{

struct node \*newnode,\*temp;

cin>>start->data;

temp=start;

char ch;

cin>>ch;

while(ch=='y')

{

newnode=new node;

cin>>newnode->data;

newnode->next=NULL;

temp->next=newnode;

temp=newnode;

cin>>ch;

}

return start;

}

void search(struct node\* start)

{

int x,flag=1;

cin>>x;

struct node\* current = start;

while (current != NULL)

{

if (current->data == x)

{

flag=1;

break;

}

else

{

flag=0;

}

current = current->next;

}

if(flag==1)

{

cout<<"FOUND";

}

else

{

cout<<"NOT FOUND";

}

}

**Output:**

**Program 28. Write a C++ program to get Nth node in a Linked List**

#include<math.h>

#include<iostream>

using namespace std;

struct node\* create(struct node\* start);

int GetNth(struct node\* start,int index);

struct node

{

int data;

struct node \*next;

};

int main()

{

int index;

struct node\* start;

start=new node;

start->next=NULL;

create(start);

cin>>index;

int res=GetNth(start,index);

cout<<res;

}

struct node\* create(struct node\* start)

{

struct node \*newnode,\*temp;

cin>>start->data;

temp=start;

char ch;

cin>>ch;

while(ch=='y')

{

newnode=new node;

cin>>newnode->data;

newnode->next=NULL;

temp->next=newnode;

temp=newnode;

cin>>ch;

}

return start;

}

int GetNth(struct node\* start,int index)

{

struct node\* current = start;

int count = 0;

while (current != NULL)

{

if (count == index)

return(current->data);

count++;

current = current->next;

}

}

Output:

**Program 29: Write a C++ program that counts the number of times a given int occurs in a Linked List**

#include<math.h>

#include<iostream>

using namespace std;

struct node\* create(struct node\* start);

void count(struct node\* start,int search\_for);

struct node

{

int data;

struct node \*next;

};

int main()

{

struct node\* start;

start=new node;

start->next=NULL;

create(start);

int search\_for;

cin>>search\_for;

count(start,search\_for);

}

struct node\* create(struct node\* start)

{

struct node \*newnode,\*temp;

cin>>start->data;

temp=start;

char ch;

cin>>ch;

while(ch=='y')

{

newnode=new node;

cin>>newnode->data;

newnode->next=NULL;

temp->next=newnode;

temp=newnode;

cin>>ch;

}

return start;

}

void count(struct node\* head, int search\_for)

{

struct node\* current = head;

int count = 0;

while (current != NULL)

{

if (current->data == search\_for)

count++;

current = current->next;

}

cout<< count;

}

**Output:**

**Program 30 : Reverse a linked list**

#include<iostream>

using namespace std;

void create(int);

struct node\* reverse();

void travel();

struct node

{

int data;

struct node \*next;

}\*head,\*head1;

int main()

{

int n;

cout<<"Enter the number of nodes->"<<endl;

cin>>n;

create(n);

reverse();

travel();

}

void create(int n)

{

int i;

struct node \*newnode,\*temp;

head=new node;

cin>>head->data;

head->next=NULL;

temp=head;

for(i=2;i<=n;i++)

{

newnode=new node;

cin>>newnode->data;

newnode->next=NULL;

temp->next=newnode;

temp=temp->next;

}

}

struct node\* reverse()

{

struct node \*ptr1,\*ptr2,\*ptr3;

ptr1=NULL;

ptr2=head;

ptr3=NULL;

while(ptr2!=NULL)

{

ptr3=ptr2->next;

ptr2->next=ptr1;

ptr1=ptr2;

ptr2=ptr3;

}

head=ptr1;

return head;

}

void travel()

{

struct node \*temp;

temp=head;

while(temp!=NULL)

{

cout<<temp->data;

temp=temp->next;

}

}

**Output:**

**Program 31:Detect loop in a linked list**

#include<iostream>

using namespace std;

struct node\* create(struct node\* start);

bool has\_cycle(node\* start);

void print(struct node\* start);

struct node

{

int data;

struct node \*next;

};

int main()

{

struct node\* start;

start=new node;

start->next=NULL;

create(start);

int d=has\_cycle(start);

cout<<d;

}

struct node\* create(struct node\* start)

{

struct node \*newnode,\*temp;

cin>>start->data;

temp=start;

char ch;

cin>>ch;

while(ch=='y')

{

newnode=new node;

cin>>newnode->data;

newnode->next=NULL;

temp->next=newnode;

temp=newnode;

cin>>ch;}

return start;}

bool has\_cycle(node\* start)

{

struct node\* fptr=start;

struct node\* sptr=start;

while(1)

{

if(fptr==NULL)

return 0;

fptr=fptr->next;

if(fptr==NULL)

return 0;

fptr=fptr->next;

sptr=sptr->next;

if(fptr==sptr)

return 1;

**Output:**

**Program 32 : Merge two sorted Linked Lists**

#include<math.h>

#include<iostream>

using namespace std;

struct node\* create(struct node\* start);

struct node\* create1(struct node\* start1);

struct node\* MergeLists(node \*start, node\* start1);

void print(node\* start);

struct node

{

int data;

struct node \*next;

};

int main()

{

struct node\* start;

start=new node;

start->next=NULL;

struct node\* start1;

start1=new node;

start1->next=NULL;

create(start);

create1(start1);

MergeLists(start,start1);

print(start);

}

struct node\* create(struct node\* start)

{

struct node \*newnode,\*temp;

cin>>start->data;

temp=start;

char ch;

cin>>ch;

while(ch=='y')

{

newnode=new node;

cin>>newnode->data;

newnode->next=NULL;

temp->next=newnode;

temp=newnode;

cin>>ch;

}

return start;

}

struct node\* create1(struct node\* start1)

{

struct node \*newnode,\*temp;

cin>>start1->data;

temp=start1;

char ch;

cin>>ch;

while(ch=='y')

{

newnode=new node;

cin>>newnode->data;

newnode->next=NULL;

temp->next=newnode;

temp=newnode;

cin>>ch;

}

return start1;

}

node\* MergeLists(node \*start, node\* start1)

{

struct node\* res=NULL;

if(!start)

{

return start1;

}

if(!start1)

{

return start;

}

if(start->data < start1->data)

{

res=start;

res->next = MergeLists(start->next,start1);

}

else

{

res=start1;

res->next = MergeLists(start,start1->next);

}

return res;

}

void print(struct node\* res)

{

struct node\* ptr=res;

while(ptr!=NULL)

{

cout<<ptr->data;

ptr=ptr->next;

}

}

**Output:**

**Program 33 : Function to check if a singly linked list is palindrome**

#include<math.h>

#include<iostream>

using namespace std;

struct node\* create(struct node\* start);

void check(struct node\* start);

struct node\* create(struct node\* start);

void print(struct node\* start);

void reverse(struct node\* start);

struct node

{

int data;

struct node \*next;

};

int main()

{

struct node\* start;

start=new node;

start->next=NULL;

create(start);

reverse(start);

}

struct node\* create(struct node\* start)

{

struct node \*newnode,\*temp;

cin>>start->data;

temp=start;

char ch;

cin>>ch;

while(ch=='y')

{

newnode=new node;

cin>>newnode->data;

newnode->next=NULL;

temp->next=newnode;

temp=newnode;

cin>>ch;

}

return start;

}

void reverse(struct node\* start)

{

struct node\* ptr1=start;

struct node\* ptr2=start;

int c=0;

int sum;

while(ptr1!=NULL)

{

c=c+1;

ptr1=ptr1->next;

}

int d=0;

while(ptr2!=NULL)

{

sum=ptr2->data\*pow(10,c-1);

d=d+sum;

ptr2=ptr2->next;

c--;

}

int t=d;

int rev=0,rem;

while(t>0)

{

rem=t%10;

rev=rev\*10+rem;

t=t/10;

}

if(rev==d)

{

cout<<"YES";

}

else

{

cout<<"NO";}}

**Output:**

**Program 34. Quick Sort with arrays**

#include<iostream>

using namespace std;

int partition(int\*,int,int);

void swap(int\*,int\*);

int partition(int\*,int,int);

void swap(int \*a,int \*b)

{

int temp;

temp=\*a;

\*a=\*b;

\*b=temp;

}

int partition (int arr[], int low, int high)

{

int pivot = arr[high];

int i = (low - 1);

for (int j = low; j <= high- 1; j++)

{

if (arr[j] <= pivot)

{

i++;

swap(&arr[i], &arr[j]);

}

}

swap(&arr[i + 1], &arr[high]);

return (i + 1);

}

void quickSort(int arr[], int low, int high)

{

if (low < high)

{

int pi = partition(arr, low, high);

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}

int main()

{

int n,i;

cin>>n;

int arr[n];

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

{

cin>>arr[i];

}

quickSort(arr, 0, n-1);

cout<<"----"<<endl;

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

{

cout<<arr[i]<<endl;

}

return 0;

}

**Output:**

**Program 35 . Merge Sort with arrays**

#include<iostream>

using namespace std;

void merge(int arr[], int l, int m, int r)

{

int i, j, k;

int n1 = m - l + 1;

int n2 = r - m;

int L[n1], R[n2];

for (i = 0; i < n1; i++)

L[i] = arr[l + i];

for (j = 0; j < n2; j++)

R[j] = arr[m + 1+ j];

i = 0;

j = 0;

k = l;

while (i < n1 && j < n2)

{

if (L[i] <= R[j])

{

arr[k] = L[i];

i++;

}

else

{

arr[k] = R[j];

j++;

}

k++;

}

while (i < n1)

{

arr[k] = L[i];

i++;

k++;

}

while (j < n2)

{

arr[k] = R[j];

j++;

k++;

}

}

void mergeSort(int arr[], int l, int r)

{

if (l < r)

{

int m = l+(r-l)/2;

mergeSort(arr, l, m);

mergeSort(arr, m+1, r);99

merge(arr, l, m, r);

}

}

void printArray(int A[], int size)

{

int i;

for (i=0; i < size; i++)

{

cout<<A[i]<<endl;

}

}

int main()

{

int arr[100],i;

int arr\_size;

cin>>arr\_size;

for(i=0;i<arr\_size;i++)

{

cin>>arr[i];

}

mergeSort(arr, 0, arr\_size - 1);

cout<<"-----"<<endl;

printArray(arr, arr\_size);

return 0;

}

**Output**

**Program 36 : Insertion Sort with arrays**

#include<iostream>

using namespace std;

void insertionSort(int arr[], int n)

{

int i, key, j;

for (i = 1; i < n; i++)

{

key = arr[i];

j = i-1;

while (j >= 0 && arr[j] > key)

{

arr[j+1] = arr[j];

j = j-1;

}

arr[j+1] = key;

}

}

void printArray(int arr[], int n)

{

int i;

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

cout<<arr[i]<<" ";

}

int main()

{

int n,i;

cin>>n;

int arr[n];

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

{

cin>>arr[i];

}

insertionSort(arr, n);

printArray(arr, n);

return 0;

}

**Output:**

**Program 37 .Selection Sort with Arrays.**

#include<iostream>

using namespace std;

void swap(int \*xp, int \*yp)

{

int temp = \*xp;

\*xp = \*yp;

\*yp = temp;

}

void selectionSort(int arr[], int n)

{

int i, j, min\_idx;

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

{

min\_idx = i;

for (j = i+1; j < n; j++)

if (arr[j] < arr[min\_idx])

min\_idx = j;

swap(&arr[min\_idx], &arr[i]);

}

}

void printArray(int arr[], int size)

{

int i;

for (i=0; i < size; i++)

cout<<arr[i]<<" ";

}

int main()

{

int n,i;

cin>>n;

int arr[n];

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

{

cin>>arr[i];

}

selectionSort(arr, n);

printf("Sorted array: \n");

printArray(arr, n);

return 0;

}

**Output:**

**Program 38: Radix Sort with Arrays.**

#include<iostream>

using namespace std;

int getMax(int arr[], int n)

{

int mx = arr[0];

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

if (arr[i] > mx)

mx = arr[i];

return mx;

}

void countSort(int arr[], int n, int exp)

{

int output[n];

int i, count[10] = {0};

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

count[ (arr[i]/exp)%10 ]++;

for (i = 1; i < 10; i++)

count[i] += count[i - 1];

for (i = n - 1; i >= 0; i--)

{

output[count[ (arr[i]/exp)%10 ] - 1] = arr[i];

count[ (arr[i]/exp)%10 ]--;

}

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

arr[i] = output[i];

}

void radixsort(int arr[], int n){

int m = getMax(arr, n);

for (int exp = 1; m/exp > 0; exp \*= 10)

countSort(arr, n, exp);

}

void print(int arr[], int n)

{

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

cout << arr[i] << " ";

}

int main()

{

int n,i;

cin>>n;

int arr[n];

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

{

cin>>arr[i];

}

radixsort(arr, n);

print(arr, n);

return 0;

}

Output:

**Program 39: Bubble Sort with Arrays**

#include<iostream>

using namespace std;

void swap(int \*xp, int \*yp)

{

int temp = \*xp;

\*xp = \*yp;

\*yp = temp;

}

void bubbleSort(int arr[], int n)

{

int i, j;

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

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

if (arr[j] > arr[j+1])

swap(&arr[j], &arr[j+1]);

}

void printArray(int arr[], int size)

{

int i;

for (i=0; i < size; i++)

cout<<arr[i]<<" ";

}

int main()

{

int n,i;

cin>>n;

int arr[n];

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

{

cin>>arr[i];

}

bubbleSort(arr, n);

printf("Sorted array: \n");

printArray(arr, n);

return 0;

}

**Output:**

**Program 40 : Program to check whether the given expression is balanced or not.**

#include<iostream>

using namespace std;

string isBalanced(string s)

{

int i=0;

int j=0;

char st[1000];

for(i=0;i<=s.length()-1;i++)

{

if(s[i]=='(' || s[i]=='{' || s[i]=='[')

{

st[j++]=s[i];

}

else

{

if(s[i]==')'&&st[--j]=='(');

else

{

if(s[i]=='}'&&st[--j]=='{');

else

{

if(s[i]==']'&&st[--j]=='[');

else

{

return "NO";

}

}

}

}

}

if(j==0)

{

return "YES";

}

else

{

return "NO";

}

}

int main()

{

string s;

cin >> s;

string result = isBalanced(s);

cout << result << endl;

return 0;}

Output:

**Program 40 : Infix to postfix conversion using stacks**

#include<iostream>

#include <cstring>

using namespace std;

#define SIZE 100000

int top=-1,k=0;

char stack[SIZE];

char input[SIZE];

char output[SIZE];

void add\_to\_output(char c);

void push(char c);

void pop();

int precedence(char c);

void display();

int main()

{

int ps,pi,i;

cin>>input;

int l=strlen(input);

input[l]=')';

l++;

cout<<"Input : "<<input;

push('(');

for(i=0;i<l;i++)

{

if(isalpha(input[i]))

{

add\_to\_output(input[i]);

}

else if(input[i]=='(')

{

push(input[i]);

}

else if(input[i]==')')

{

while(stack[top]!='(')

{

add\_to\_output(stack[top]);

pop();

}

pop();

}

else

{

if(stack[top]=='(')

{

push(input[i]);

}

else

{

pi=precedence(input[i]);

ps=precedence(stack[top]);

if(ps>=pi)

{

while(ps>=pi)

{

add\_to\_output(stack[top]);

pop();

if(stack[top]=='(' || stack[top]==')')

break;

ps=precedence(stack[top]);

}

push(input[i]);

}

else

{

push(input[i]);

}

}

}

}

display();

}

void push(char c)

{

if(top+1==SIZE)

{

cout<<"Stack Overflow Condition";

}

top=top+1;

stack[top]=c;

}

void add\_to\_output(char c)

{

if(c!='(' && c!=')')

output[k]=c;

k++;

}

void pop()

{

if(top==-1)

{

cout<<endl<<"Stack Underflow Condition";

}

top=top-1;

}

int precedence(char c)

{

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

{

return 1;

}

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

{

return 2;

}

else if(c=='^')

{

return 3;

}

}

void display()

{

cout<<endl<<"Output = "<<output;

}

**Output:**

**Program 41: Infix to prefix conversion using stacks**

#include<iostream>

#include <cstring>

using namespace std;

#define SIZE 100000

int top=-1,k=0;

char stack[SIZE];

char input[SIZE];

char output[SIZE];

void add\_to\_output(char c);

void push(char c);

void pop();

int precedence(char c);

void display();

void swap(char \*a,char \*b);

int main()

{

int ps,pi,i,m;

cin>>input;

int l=strlen(input);

cout<<"Input : "<<input;

m=l-1;

for(i=0;i<l/2;i++)

{

swap(&input[i],&input[m]);

m--;

}

input[l++]='(';

push(')');

for(i=0;i<l;i++)

{

if(isalpha(input[i]))

{

add\_to\_output(input[i]);

}

else if(input[i]==')')

{

push(input[i]);

}

else if(input[i]=='(')

{

while(stack[top]!=')')

{

add\_to\_output(stack[top]);

pop();

}

pop();

}

else

{

if(stack[top]==')')

{

push(input[i]);

}

else

{

ps=precedence(stack[top]);

pi=precedence(input[i]);

if(ps>=pi)

{

while(ps>=pi)

{

add\_to\_output(stack[top]);

pop();

if(stack[top]==')' || stack[top]=='(')

{

break;

}

ps=precedence(stack[top]);

}

push(input[i]);

}

else

{

push(input[i]);

}}}}

l=strlen(output);

m=l-1;

for(int j=0;j<l/2;j++)

{

swap(&output[j],&output[m]);

m--;

}

display();

}

void swap(char \*a,char \*b)

{

char temp=\*a;

\*a=\*b;

\*b=temp;

}

void push(char c)

{

if(top+1==SIZE)

{

cout<<"Stack Overflow Condition";

}

top=top+1;

stack[top]=c;

}

void add\_to\_output(char c)

{

if(c!='(' && c!=')')

output[k]=c;

k++;

}

void pop()

{

if(top==-1)

{

cout<<endl<<"Stack Underflow Condition";

}top=top-1;

}

int precedence(char c)

{

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

{

return 1;

}

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

{

return 2;

}

else if(c=='^')

{

return 3;

}

}

void display()

{

cout<<endl<<"Output = "<<output;

}

**Program 42: Evaluation of prefix expression using stack**

#include<iostream>

#include<cstring>

using namespace std;

int stack[10];

int top=-1;

int pop();

void push(int);

int get\_type(char c );

int main()

{

char str[10];

int len,op1,op2,val,value,i;

cout<<"Enter the exp";

cin>>str;

len=strlen(str);

for(i=len-1;i>=0;i--)

{

switch (get\_type(str[i]))

{

case 0:

val=str[i]-'0';

push(val);

break;

case 1:

op1=pop();

op2=pop();

switch(str[i])

{

case '+':

value=op1 + op2;

break;

case '-':

value=op1 - op2;

break;

case '\*':

value=op1 \* op2;

break;

case '/':

value=op1 / op2;

break;

}

push(value);

}

}

cout<<"RES->"<<stack[0];

}

void push(int val)

{

top++;

stack[top]=val;

}

int pop()

{

return(stack[top--]);

}

int get\_type(char c )

{

if(c=='+' || c=='-' || c=='\*' || c=='/')

return 1;

else

return 0;

}

**Output:**

**Program 43: Evaluation of postfix expression using stack**

#include<iostream>

#include<string>

#define MAX 100

using namespace std;

float stack[MAX];

int top=-1;

void push(float stack[],float val);

float pop(float stack[]);

float eval(char str[]);

int main()

{

float val;

char str[100];

cout<<"Enter the exp";

cin>>str;

val=eval(str);

cout<<val;

}

void push(float stack[],float val)

{

if(top+1==MAX)

{

cout<<"Overflow";

}

else

{

top++;

stack[top]=val;

}

}

float pop(float stack[])

{

float val=-1;

if(top==-1)

{

cout<<"Stack underflow";

}

else

{

val=stack[top];

top--;

}

return val;

}

float eval(char str[])

{

int i=0;

float op1,op2,value;

while(str[i]!='\0')

{

if(isdigit(str[i]))

{

push(stack,(float)(str[i]-'0'));

}

else

{

op1=pop(stack);

op2=pop(stack);

switch(str[i])

{

case '+':

value=op1 + op2;

break;

case '-':

value=op1 - op2;

break;

case '\*':

value=op1 \* op2;

break;

case '/':

value=op1 / op2;

break;

}

push(stack,value);

}

i++;

}

return(pop(stack));

}

**Output:**

**Program 44: Merge two sorted linked lists**

#include<math.h>

#include<iostream>

using namespace std;

struct node\* create(struct node\* start);

struct node\* create1(struct node\* start1);

struct node\* MergeLists(node \*start, node\* start1);

void print(node\* start);

struct node

{

int data;

struct node \*next;

};

int main()

{

struct node\* start;

start=new node;

start->next=NULL;

struct node\* start1;

start1=new node;

start1->next=NULL;

create(start);

create1(start1);

MergeLists(start,start1);

print(start);

}

struct node\* create(struct node\* start)

{

struct node \*newnode,\*temp;

cin>>start->data;

temp=start;

char ch;

cin>>ch;

while(ch=='y')

{

newnode=new node;

cin>>newnode->data;

newnode->next=NULL;

temp->next=newnode;

temp=newnode;

cin>>ch;

}

return start;

}

struct node\* create1(struct node\* start1)

{

struct node \*newnode,\*temp;

cin>>start1->data;

temp=start1;

char ch;

cin>>ch;

while(ch=='y')

{

newnode=new node;

cin>>newnode->data;

newnode->next=NULL;

temp->next=newnode;

temp=newnode;

cin>>ch;

}

return start1;

}

node\* MergeLists(node \*start, node\* start1)

{

struct node\* res=NULL;

if(!start)

{

return start1;

}

if(!start1)

{

return start;

}

if(start->data < start1->data)

{

res=start;

res->next = MergeLists(start->next,start1);

}

else

{

res=start1;

res->next = MergeLists(start,start1->next);

}

return res;

}

void print(struct node\* res)

{

struct node\* ptr=res;

while(ptr!=NULL)

{

cout<<ptr->data;

ptr=ptr->next;

}

}

**Output:**

**Program 45: Write a C++ program to print the middle of a given linked list**

#include<math.h>

#include<iostream>

using namespace std;

void printMiddle(struct node \*start);

struct node\* create(struct node\* start);

void print(struct node\* start);

struct node

{

int data;

struct node \*next;

};

int main()

{

struct node\* start;

start=new node;

start->next=NULL;

create(start);

printMiddle(start);

}

struct node\* create(struct node\* start)

{

struct node \*newnode,\*temp;

cin>>start->data;

temp=start;

char ch;

cin>>ch;

while(ch=='y')

{

newnode=new node;

cin>>newnode->data;

newnode->next=NULL;

temp->next=newnode;

temp=newnode;

cin>>ch;

}

return start;

}

void printMiddle(struct node \*start)

{

struct node \*slow\_ptr = start;

struct node \*fast\_ptr = start;

if (start!=NULL)

{

while (fast\_ptr != NULL && fast\_ptr->next != NULL)

{

fast\_ptr = fast\_ptr->next->next;

slow\_ptr = slow\_ptr->next;

}

cout<<"The middle element is"<<slow\_ptr->data;

}

}

Output:

**Program 46: Implement queues using stack.**

#include<iostream>

using namespace std;

struct sNode

{

int data;

struct sNode \*next;

};

void push(struct sNode\*\* top\_ref, int new\_data);

int pop(struct sNode\*\* top\_ref);

struct queue

{

struct sNode \*stack1;

struct sNode \*stack2;

};

void enQueue(struct queue \*q, int x)

{

push(&q->stack1, x);

}

int deQueue(struct queue \*q)

{

int x;

if(q->stack1 == NULL && q->stack2 == NULL)

{

cout<<"Q is empty";

getchar();

exit(0);

}

if(q->stack2 == NULL)

{

while(q->stack1 != NULL)

{

x = pop(&q->stack1);

push(&q->stack2, x);

}

}

x = pop(&q->stack2);

return x;

}

void push(struct sNode\*\* top\_ref, int new\_data)

{

struct sNode\* new\_node = new sNode;

if(new\_node == NULL)

{

cout<<"Stack overflow \n";

getchar();

exit(0);

}

new\_node->data = new\_data;

new\_node->next = (\*top\_ref);

(\*top\_ref) = new\_node;

}

int pop(struct sNode\*\* top\_ref)

{

int res;

struct sNode \*top;

if(\*top\_ref == NULL)

{

cout<<"Stack overflow \n";

getchar();

exit(0);

}

else

{

top = \*top\_ref;

res = top->data;

\*top\_ref = top->next;

delete(top);

return res;

}

}

int main()

{

struct queue \*q = new queue;

q->stack1 = NULL;

q->stack2 = NULL;

enQueue(q, 1);

enQueue(q, 2);

enQueue(q, 3);

cout<<deQueue(q);

cout<<deQueue(q);

cout<<deQueue(q);

}

Output:

**Program 47: Implement stacks using queues.**

#include<iostream>

using namespace std;

class Stack

{

queue<int> q1, q2;

int curr\_size;

public:

Stack()

{

curr\_size = 0;

}

void pop()

{

if (q1.empty())

return;

while (q1.size() != 1)

{

q2.push(q1.front());

q1.pop();

}

q1.pop();

curr\_size--;

queue<int> q = q1;

q1 = q2;

q2 = q;

}

void push(int x)

{

q1.push(x);

curr\_size++;

}

int top()

{

if (q1.empty())

return -1;

while( q1.size() != 1 )

{

q2.push(q1.front());

q1.pop();

}

int temp = q1.front();

q2.push(temp);

queue<int> q = q1;

q1 = q2;

q2 = q;

return temp;

}

int size()

{

return curr\_size;

}

};

int main()

{

Stack s;

s.push(1);

s.push(2);

s.push(3);

s.push(4);

cout << "current size: " << s.size()

<< endl;

cout << s.top() << endl;

s.pop();

cout << s.top() << endl;

s.pop();

cout << s.top() << endl;

cout << "current size: " << s.size()

<< endl;

return 0;

}

**Ouptut:**

**Program 48 : C++ program to create a file, write and read text in/from file.**

#include<iostream>

#include<fstream>

using namespace std;

int main()

{

fstream outf("item.txt");

cout<<"Enter item name";

char name[30];

cin>>name;

outf<<name<<"\n";

cout<<"Enter item cost";

float cost;

cin>>cost;

outf<<cost<<"\n";

outf.close();

fstream inf("item.txt");

inf>>name;

inf>>cost;

cout<<"\n";

cout<<"Item name:"<<name<<"\n";

cout<<"Item cost:"<<cost<<"\n";

inf.close();

return 0;

}

**Output:**

**Prorgam 49 : Perform Push, Pop operations in stacks with linked list implementation**

#include<iostream>

using namespace std;

struct StackNode

{

int data;

struct StackNode\* next;

};

struct StackNode\* newNode(int data)

{

struct StackNode\* stackNode = new StackNode;

stackNode->data = data;

stackNode->next = NULL;

return stackNode;

}

int isEmpty(struct StackNode \*root)

{

return !root;

}

void push(struct StackNode\*\* root, int data)

{

struct StackNode\* stackNode = newNode(data);

stackNode->next = \*root;

\*root = stackNode;

cout<<"pushed to stack\n"<<data;

}

int pop(struct StackNode\*\* root)

{

if (isEmpty(\*root))

return INT\_MIN;

struct StackNode\* temp = \*root;

\*root = (\*root)->next;

int popped = temp->data;

delete temp;

return popped;

}

int peek(struct StackNode\* root)

{

if (isEmpty(root))

return INT\_MIN;

return root->data;

}

int main()

{

struct StackNode\* root = NULL;

push(&root, 10);

push(&root, 20);

push(&root, 30);

cout<<" popped from stack\n"<<pop(&root);

cout<<"Top element is \n"<<peek(root);

return 0;

}

**Output:**

**Program 50 : Insertion and deletion in linear queue**

#include <iostream>

using namespace std;

#define MAX 5

class Queue

{

private:

int front,rear;

int ele[MAX];

public:

Queue()

{

front = 0;

rear = -1;

}

int isFull();

int isEmpty();

void insertQueue(int item);

int deleteQueue(int \*item);

};

int Queue::isFull()

{

int full = 0 ;

if( rear == MAX-1 )

full = 1;

return full;

}

int Queue::isEmpty()

{

int empty = 0 ;

if( front == rear + 1 )

empty = 1;

return empty;

}

void Queue:: insertQueue(int item)

{

if( isFull() )

{

cout<<"\nQueue OverFlow" << endl;

return;

}

ele[++rear]=item;

cout<<"\ninserted Value :" << item;

}

int Queue:: deleteQueue(int \*item)

{

if( isEmpty() )

{

cout<<"\nQueue Underflow" << endl;

return -1;

}

\*item = ele[front++];

return 0;

}

int main()

{

int item=0;

Queue q = Queue();

q.insertQueue(10);

q.insertQueue(20);

q.insertQueue(30);

q.insertQueue(40);

q.insertQueue(50);

q.insertQueue(60);

if(q.deleteQueue( &item)==0)

cout<<"\nDeleted item : "<< item;

if(q.deleteQueue( &item)==0)

cout<<"\nDeleted item : "<< item;

if(q.deleteQueue( &item)==0)

cout<<"\nDeleted item : "<< item;

if(q.deleteQueue( &item)==0)

cout<<"\nDeleted item : "<< item;

if(q.deleteQueue( &item)==0)

cout<<"\nDeleted item : "<< item;

if(q.deleteQueue( &item)==0)

cout<<"\nDeleted item : "<< item;

cout<< endl;

return 0;

}

Output:

**Program 51 : Insertion and deletion in circular queue**

#include<bits/stdc++.h>

using namespace std;

struct Queue

{

int rear, front;

int size;

int \*arr;

Queue(int s)

{

front = rear = -1;

size = s;

arr = new int[s];

}

void enQueue(int value);

int deQueue();

void displayQueue();

};

void Queue::enQueue(int value)

{

if ((front == 0 && rear == size-1) ||

(rear == (front-1)%(size-1)))

{

cout<<"Queue is Full"<<endl;

return;

}

else if (front == -1)

{

front = rear = 0;

arr[rear] = value;

}

else if (rear == size-1 && front != 0)

{

rear = 0;

arr[rear] = value;

}

else

{

rear++;

arr[rear] = value;

}

}

int Queue::deQueue()

{

if (front == -1)

{

cout<<"Queue is Empty"<<endl;

return INT\_MIN;

}

int data = arr[front];

arr[front] = -1;

if (front == rear)

{

front = -1;

rear = -1;

}

else if (front == size-1)

front = 0;

else

front++;

return data;

}

void Queue::displayQueue()

{

if (front == -1)

{

cout<<"Queue is Empty"<<endl;

return;

}

cout<<"Elements in Circular Queue are: "<<endl;

if (rear >= front)

{

for (int i = front; i <= rear; i++)

cout<<arr[i];

}

else

{

for (int i = front; i < size; i++)

cout<<arr[i];

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

cout<<arr[i];

}

}

int main()

{

Queue q(5);

q.enQueue(14);

q.enQueue(22);

q.enQueue(13);

q.enQueue(-6);

q.displayQueue();

cout<<"Deleted value = "<<q.deQueue()<<endl;

cout<<"Deleted value = "<<q.deQueue()<<endl;

q.displayQueue();

q.enQueue(9);

q.enQueue(20);

q.enQueue(5);

q.displayQueue();

q.enQueue(20);

return 0;

}

**Output:**

**Program 52 : Perform Push, Pop operations in stacks with array implementation**

#include<bits/stdc++.h>

using namespace std;

#define MAX 1000

class Stack

{

int top;

public:

int a[MAX];

Stack() { top = -1; }

bool push(int x);

int pop();

bool isEmpty();

};

bool Stack::push(int x)

{

if (top >= MAX)

{

cout << "Stack Overflow";

return false;

}

else

{

a[++top] = x;

return true;

}

}

int Stack::pop()

{

if (top < 0)

{

cout << "Stack Underflow";

return 0;

}

else

{

int x = a[top--];

return x;

}

}

bool Stack::isEmpty()

{

return (top < 0);

}

int main()

{

struct Stack s;

s.push(10);

s.push(20);

s.push(30);

cout << s.pop() << " Popped from stack\n";

return 0;

}

**Output:**

**Program 53 : Insertion in Circular linked list at a given position (begin, end, middle)**

#include<iostream>

using namespace std;

struct node\* create();

void travel();

struct node\* insert();

struct node

{

int data;

struct node \*next;

}\*head;

int main()

{

cout<<"SEC->112"<<endl;

create();

insert();

travel();

}

struct node\* create()

{

int i;

char ch;

struct node \*newnode,\*temp;

head=new node;

cin>>head->data;

head->next=NULL;

temp=head;

cin>>ch;

while(ch=='y')

{

newnode=new node;

cin>>newnode->data;

newnode->next=NULL;

temp->next=newnode;

temp=newnode;

cin>>ch;

}

temp->next=head;

return head;

}

struct node\* insert()

{

int d;

cin>>d;

struct node \*newnode,\*ptr1;

ptr1=head;

newnode=new node;

newnode->data=d;

newnode->next=head;

while(ptr1->next!=head)

{

ptr1=ptr1->next;

}

ptr1->next=newnode;

return head;

}

void travel()

{

struct node \*current=head;

do

{

cout<<current->data;

current = current->next;

}while(current != head);

}

**Output:**

**Program 54 : Deletion in Double linked list at a given position (begin, end, middle)**

#include<iostream>

using namespace std;

struct node\* create(struct node\* start);

struct node\* insert(struct node\* start);

void print(struct node\* start);

struct node

{

int data;

struct node \*next,\*prev;

};

int main()

{

struct node\* start;

start=new node;

start->next=NULL;

start->prev=NULL;

create(start);

insert(start);

print(start);

}

struct node\* create(struct node\* start)

{

struct node \*newnode,\*temp;

cin>>start->data;

temp=start;

char ch;

cin>>ch;

while(ch=='y')

{

newnode=new node;

cin>>newnode->data;

newnode->next=NULL;

newnode->prev=temp;

temp->next=newnode;

temp=newnode;

cin>>ch;

}

return start;

}

struct node\* insert(struct node\* start)

{

int d;

cout<<"Enter the value to be deleted"<<endl;

cin>>d;

struct node\* ptr1,\*ptr2;

ptr1=start;

while(ptr1->next->data!=d)

{

ptr1=ptr1->next;

}

ptr2=ptr1->next;

ptr1->next=ptr1->next->next;

delete ptr2;

return start;

}

void print(struct node\* start)

{

struct node\* ptr1=start;

while(ptr1!=NULL)

{

cout<<ptr1->data;

ptr1=ptr1->next;

}

}

**Output:**

**Program 55 : Insertion in Double linked list at a given position (begin, end, middle)**

#include<iostream>

using namespace std;

struct node\* create(struct node\* start);

void print(struct node\* start);

struct node\* insert(struct node\* start);

struct node

{

int data;

struct node \*next,\*prev;

};

int main()

{

struct node\* start;

start=new node;

start->next=NULL;

start->prev=NULL;

create(start);

insert(start);

print(start);

}

struct node\* create(struct node\* start)

{

struct node \*newnode,\*temp;

cin>>start->data;

temp=start;

char ch;

cin>>ch;

while(ch=='y')

{

newnode=new node;

cin>>newnode->data;

newnode->next=NULL;

newnode->prev=temp;

temp->next=newnode;

temp=newnode;

cin>>ch;

}

return start;

}

struct node\* insert(struct node\* start)

{

int pos;

cout<<"pos";

cin>>pos;

struct node\* newnode,\*ptr;

ptr=start;

newnode=new node;

cin>>newnode->data;

int c=0;

while(pos> ++c)

{

ptr=ptr->next;

}

newnode->next=ptr->next;

newnode->prev=ptr;

ptr->next=newnode;

return start;

}

void print(struct node\* start)

{

struct node\* ptr1=start;

while(ptr1!=NULL)

{

cout<<ptr1->data;

ptr1=ptr1->next;

}

}

**Output:**

**Program 56 : Intersection point of two Linked Lists.**

#include<iostream>

using namespace std;

struct Node

{

int data;

struct Node\* next;

};

int getCount(struct Node\* head);

int \_getIntesectionNode(int d, struct Node\* head1, struct Node\* head2);

int getIntesectionNode(struct Node\* head1, struct Node\* head2)

{

int c1 = getCount(head1);

int c2 = getCount(head2);

int d;

if(c1 > c2)

{

d = c1 - c2;

return \_getIntesectionNode(d, head1, head2);

}

else

{

d = c2 - c1;

return \_getIntesectionNode(d, head2, head1);

}

}

int \_getIntesectionNode(int d, struct Node\* head1, struct Node\* head2)

{

int i;

struct Node\* current1 = head1;

struct Node\* current2 = head2;

for(i = 0; i < d; i++)

{

if(current1 == NULL)

{ return -1; }

current1 = current1->next;

}

while(current1 != NULL && current2 != NULL)

{

if(current1 == current2)

return current1->data;

current1= current1->next;

current2= current2->next;

}

return -1;

}

int getCount(struct Node\* head)

{

struct Node\* current = head;

int count = 0;

while (current != NULL)

{

count++;

current = current->next;

}

return count;

}

int main()

{

struct Node\* newNode;

struct Node\* head1=new Node;

head1->data = 10;

struct Node\* head2 = new Node;

head2->data = 3;

newNode = new Node;

newNode->data = 6;

head2->next = newNode;

newNode = new Node;

newNode->data = 9;

head2->next->next = newNode;

newNode = new Node;

newNode->data = 15;

head1->next = newNode;

head2->next->next->next = newNode;

newNode = new Node;

newNode->data = 30;

head1->next->next= newNode;

head1->next->next->next = NULL;

cout<<"\n The Node of intersection is\n"<<getIntesectionNode(head1, head2);

}

**Output:**

**Program 57 : Remove duplicates from a sorted linked list.**

#include<iostream>

using namespace std;

struct Node

{

int data;

struct Node\* next;

};

void removeDuplicates(struct Node\* head)

{

struct Node\* current = head;

struct Node\* next\_next;

if (current == NULL)

return;

while (current->next != NULL)

{

if (current->data == current->next->data)

{

next\_next = current->next->next;

delete(current->next);

current->next = next\_next;

}

else

{

current = current->next;

}

}

}

void push(struct Node\*\* head\_ref, int new\_data)

{

struct Node\* new\_Node = new Node;

new\_Node->data = new\_data;

new\_Node->next = (\*head\_ref);

(\*head\_ref) = new\_Node;

}

void printList(struct Node \*Node)

{

while (Node!=NULL)

{

cout<<"%d ", Node->data;

Node = Node->next;

}

}

int main()

{

struct Node\* head = NULL;

push(&head, 20);

push(&head, 13);

push(&head, 13);

push(&head, 11);

push(&head, 11);

push(&head, 11);

cout<<"\n Linked list before duplicate removal";

printList(head);

removeDuplicates(head);

cout<<"\n Linked list after duplicate removal";

printList(head);

return 0;

}

**Output:**

**Program 58 .Remove duplicates from an unsorted linked list**

#include<iostream>

using namespace std;

struct Node

{

int data;

struct Node \*next;

};

struct Node \*newNode(int data)

{

Node \*temp = new Node;

temp->data = data;

temp->next = NULL;

return temp;

}

void removeDuplicates(struct Node \*start)

{

struct Node \*ptr1, \*ptr2, \*dup;

ptr1 = start;

while (ptr1 != NULL && ptr1->next != NULL)

{

ptr2 = ptr1;

while (ptr2->next != NULL)

{

if (ptr1->data == ptr2->next->data)

{

dup = ptr2->next;

ptr2->next = ptr2->next->next;

delete(dup);

}

else

ptr2 = ptr2->next;

}

ptr1 = ptr1->next;

}

}

void printList(struct Node \*Node)

{

while (Node != NULL)

{

cout<<Node->data;

Node = Node->next;

}

}

int main()

{

struct Node \*start = newNode(10);

start->next = newNode(12);

start->next->next = newNode(11);

start->next->next->next = newNode(11);

start->next->next->next->next = newNode(12);

start->next->next->next->next->next =

newNode(11);

start->next->next->next->next->next->next =

newNode(10);

cout<<"Linked list before removing duplicates ";

printList(start);

removeDuplicates(start);

cout<<"\nLinked list after removing duplicates ";

printList(start);

return 0;

}

**Output:**

**Program 59 : Pairwise swap elements of a given linked list**

#include<iostream>

using namespace std;

struct Node

{

int data;

struct Node \*next;

};

void pairWiseSwap(struct Node \*\*head)

{

if (\*head == NULL || (\*head)->next == NULL)

return;

struct Node \*prev = \*head;

struct Node \*curr = (\*head)->next;

\*head = curr;

while (true)

{

struct Node \*next = curr->next;

curr->next = prev;

if (next == NULL || next->next == NULL)

{

prev->next = next;

break;

}

prev->next = next->next;

prev = next;

curr = prev->next;

}

}

void push(struct Node\*\* head\_ref, int new\_data)

{

struct Node\* new\_Node = new Node;

new\_Node->data = new\_data;

new\_Node->next = (\*head\_ref);

(\*head\_ref) = new\_Node;

}

void printList(struct Node \*Node)

{

while(Node != NULL)

{

cout<<Node->data;

Node = Node->next;

}

}

int main()

{

struct Node \*start = NULL;

push(&start, 7);

push(&start, 6);

push(&start, 5);

push(&start, 4);

push(&start, 3);

push(&start, 2);

push(&start, 1);

cout<<"\n Linked list before calling pairWiseSwap() ";

printList(start);

pairWiseSwap(&start);

cout<<"\n Linked list after calling pairWiseSwap() ";

printList(start);

return 0;

}

**Output:**

**Program 60 : Move last element to front of a given Linked List**

#include<iostream>

using namespace std;

struct Node

{

int data;

struct Node \*next;

};

void moveToFront(struct Node \*\*head\_ref)

{

if (\*head\_ref == NULL || (\*head\_ref)->next == NULL)

return;

struct Node \*secLast = NULL;

struct Node \*last = \*head\_ref;

while (last->next != NULL)

{

secLast = last;

last = last->next;

}

secLast->next = NULL;

last->next = \*head\_ref;

\*head\_ref = last;

}

void push(struct Node\*\* head\_ref, int new\_data)

{

struct Node\* new\_Node = new Node;

new\_Node->data = new\_data;

new\_Node->next = (\*head\_ref);

(\*head\_ref) = new\_Node;

}

void printList(struct Node \*Node)

{

while(Node != NULL)

{

cout<< Node->data;

Node = Node->next;

}

}

int main()

{

struct Node \*start = NULL;

push(&start, 5);

push(&start, 4);

push(&start, 3);

push(&start, 2);

push(&start, 1);

cout<<"\n Linked list before moving last to front\n";

printList(start);

moveToFront(&start);

cout<<"\n Linked list after removing last to front\n";

printList(start);

return 0;

}

**Output:**

**Program 61: Segregate even and odd nodes in a Linked List**

#include<iostream>

using namespace std;

struct Node

{

int data;

struct Node \*next;

};

void segregateEvenOdd(struct Node \*\*head\_ref)

{

struct Node \*end = \*head\_ref;

struct Node \*prev = NULL;

struct Node \*curr = \*head\_ref;

while (end->next != NULL)

end = end->next;

struct Node \*new\_end = end;

while (curr->data %2 != 0 && curr != end)

{

new\_end->next = curr;

curr = curr->next;

new\_end->next->next = NULL;

new\_end = new\_end->next;

}

if (curr->data%2 == 0)

{

\*head\_ref = curr;

while (curr != end)

{

if ( (curr->data)%2 == 0 )

{

prev = curr;

curr = curr->next;

}

else

{

prev->next = curr->next;

curr->next = NULL;

new\_end->next = curr;

new\_end = curr;

curr = prev->next;

}

}

}

else prev = curr;

if (new\_end!=end && (end->data)%2 != 0)

{

prev->next = end->next;

end->next = NULL;

new\_end->next = end;

}

return;

}

void push(struct Node\*\* head\_ref, int new\_data)

{

struct Node\* new\_node = new Node;

new\_node->data = new\_data;

new\_node->next = (\*head\_ref);

(\*head\_ref) = new\_node;

}

void printList(struct Node \*node)

{

while (node!=NULL)

{

cout<<node->data<<" ";

node = node->next;

}

}

int main()

{

struct Node\* head = NULL;

push(&head, 11);

push(&head, 20);

push(&head, 8);

push(&head, 4);

push(&head, 6);

push(&head, 2);

push(&head, 1);

cout<<"\nOriginal Linked list \n";

printList(head);

segregateEvenOdd(&head);

cout<<"\nModified Linked list \n";

printList(head);

}

**Output:**

**Program 62 : Add two numbers represented by linked lists**

#include<iostream>

#include<math.h>

using namespace std;

struct Node

{

int data;

struct Node\* next;

};

typedef struct Node Node;

void push(struct Node\*\* head\_ref, int new\_data)

{

struct Node\* new\_Node = new Node;

new\_Node->data = new\_data;

new\_Node->next = (\*head\_ref);

(\*head\_ref) = new\_Node;

}

void printList(struct Node \*Node)

{

while (Node != NULL)

{

cout<<Node->data<<" ";

Node = Node->next;

}

cout<<endl;

}

void swapPointer( Node\*\* a, Node\*\* b )

{

Node\* t = \*a;

\*a = \*b;

\*b = t;

}

int getSize(struct Node \*Node)

{

int size = 0;

while (Node != NULL)

{

Node = Node->next;

size++;

}

return size;

}

Node\* addSameSize(Node\* head1, Node\* head2, int\* carry)

{

if (head1 == NULL)

return NULL;

int sum;

Node\* result = new Node;

result->next = addSameSize(head1->next, head2->next, carry);

sum = head1->data + head2->data + \*carry;

\*carry = sum / 10;

sum = sum % 10;

result->data = sum;

return result;

}

void addCarryToRemaining(Node\* head1, Node\* cur, int\* carry, Node\*\* result)

{

int sum;

if (head1 != cur)

{

addCarryToRemaining(head1->next, cur, carry, result);

sum = head1->data + \*carry;

\*carry = sum/10;

sum %= 10;

push(result, sum);

}

}

void addList(Node\* head1, Node\* head2, Node\*\* result)

{

Node \*cur;

if (head1 == NULL)

{

\*result = head2;

return;

}

else if (head2 == NULL)

{

\*result = head1;

return;

}

int size1 = getSize(head1);

int size2 = getSize(head2) ;

int carry = 0;

if (size1 == size2)

\*result = addSameSize(head1, head2, &carry);

else

{

int diff = abs(size1 - size2);

if (size1 < size2)

swapPointer(&head1, &head2);

for (cur = head1; diff--; cur = cur->next);

\*result = addSameSize(cur, head2, &carry);

addCarryToRemaining(head1, cur, &carry, result);

}

if (carry)

push(result, carry);

}

int main()

{

Node \*head1 = NULL, \*head2 = NULL, \*result = NULL;

int arr1[] = {9, 9, 9};

int arr2[] = {1, 8};

int size1 = sizeof(arr1) / sizeof(arr1[0]);

int size2 = sizeof(arr2) / sizeof(arr2[0]);

int i;

for (i = size1-1; i >= 0; --i)

push(&head1, arr1[i]);

for (i = size2-1; i >= 0; --i)

push(&head2, arr2[i]);

addList(head1, head2, &result);

printList(result);

return 0;

}

**Output:**

**Program 63 : Subtract two numbers represented by linked lists**

#include<bits/stdc++.h>

using namespace std;

struct Node

{

int data;

struct Node\* next;

};

Node\* newNode(int data)

{

Node\* temp = new Node;

temp->data = data;

temp->next = NULL;

return temp;

}

int getLength(Node \*Node)

{

int size = 0;

while (Node != NULL)

{

Node = Node->next;

size++;

}

return size;

}

Node\* paddZeros(Node\* sNode, int diff)

{

if (sNode == NULL)

return NULL;

Node\* zHead = newNode(0);

diff--;

Node\* temp = zHead;

while (diff--)

{

temp->next = newNode(0);

temp = temp->next;

}

temp->next = sNode;

return zHead;

}

Node\* subtractLinkedListHelper(Node\* l1, Node\* l2, bool& borrow)

{

if (l1 == NULL && l2 == NULL && borrow == 0)

return NULL;

Node\* previous = subtractLinkedListHelper(l1 ? l1->next : NULL,

l2 ? l2->next : NULL, borrow);

int d1 = l1->data;

int d2 = l2->data;

int sub = 0;

if (borrow)

{

d1--;

borrow = false;

}

if (d1 < d2)

{

borrow = true;

d1 = d1 + 10;

}

sub = d1 - d2;

Node\* current = newNode(sub);

current->next = previous;

return current;

}

Node\* subtractLinkedList(Node\* l1, Node\* l2)

{

if (l1 == NULL && l2 == NULL)

return NULL;

int len1 = getLength(l1);

int len2 = getLength(l2);

Node \*lNode = NULL, \*sNode = NULL;

Node\* temp1 = l1;

Node\* temp2 = l2;

if (len1 != len2)

{

lNode = len1 > len2 ? l1 : l2;

sNode = len1 > len2 ? l2 : l1;

sNode = paddZeros(sNode, abs(len1 - len2));

}

else

{

while (l1 && l2)

{

if (l1->data != l2->data)

{

lNode = l1->data > l2->data ? temp1 : temp2;

sNode = l1->data > l2->data ? temp2 : temp1;

break;

}

l1 = l1->next;

l2 = l2->next;

}

}

bool borrow = false;

return subtractLinkedListHelper(lNode, sNode, borrow);

}

void printList(struct Node \*Node)

{

while (Node != NULL)

{

cout<<Node->data<<" ";

Node = Node->next;

}

}

int main()

{

Node\* head1 = newNode(9);

head1->next = newNode(11);

head1->next->next = newNode(12);

Node\* head2 = newNode(10);

Node\* result = subtractLinkedList(head1, head2);

printList(result);

return 0;

}

**Output:**

**Program 64 : Multiply two numbers represented by linked lists.**

#include<iostream>

using namespace std;

struct node

{

int data;

struct node\* next;

};

struct node \*newNode(int data)

{

struct node \*new\_node = new node;

new\_node->data = data;

new\_node->next = NULL;

return new\_node;

}

void push(struct node\*\* head\_ref, int new\_data)

{

struct node\* new\_node = newNode(new\_data);

new\_node->next = (\*head\_ref);

(\*head\_ref) = new\_node;

}

long multiplyTwoLists (struct node\* first, struct node\* second)

{

int num1 = 0, num2 = 0;

while (first || second)

{

if (first)

{

num1 = num1\*10 + first->data;

first = first->next;

}

if (second)

{

num2 = num2\*10 + second->data;

second = second->next;

}

}

return num1\*num2;

}

void printList(struct node \*node)

{

while(node!= NULL)

{

cout<<node->data;

node = node->next;

}

cout<<endl;

}

int main(void)

{

struct node\* first = NULL;

struct node\* second = NULL;

push(&first, 6);

push(&first, 4);

push(&first, 9);

cout<<"First List is: ";

printList(first);

// create second list 8->4

push(&second, 4);

push(&second, 8);

cout<<"Second List is: ";

printList(second);

cout<<"Result is: ";

printf("%ld", multiplyTwoLists(first, second));

return 0;

}

**Output:**

**Program 65 : Intersection of two Sorted Linked Lists**

#include<iostream>

using namespace std;

struct Node

{

int data;

struct Node\* next;

};

void push(struct Node\*\* head\_ref, int new\_data);

struct Node\* sortedIntersect(struct Node\* a, struct Node\* b)

{

struct Node dummy;

struct Node\* tail = &dummy;

dummy.next = NULL;

while (a != NULL && b != NULL)

{

if (a->data == b->data)

{

push((&tail->next), a->data);

tail = tail->next;

a = a->next;

b = b->next;

}

else if (a->data < b->data)

a = a->next;

else

b = b->next;

}

return(dummy.next);

}

void push(struct Node\*\* head\_ref, int new\_data)

{

struct Node\* new\_Node = new Node;

new\_Node->data = new\_data;

new\_Node->next = (\*head\_ref);

(\*head\_ref) = new\_Node;

}

void printList(struct Node \*Node)

{

while (Node != NULL)

{

cout<<Node->data;

Node = Node->next;

}

}

int main()

{

struct Node\* a = NULL;

struct Node\* b = NULL;

struct Node \*intersect = NULL;

push(&a, 6), push(&a, 5),push(&a, 4);

push(&a, 3);

push(&a, 2);

push(&a, 1);

push(&b, 8);

push(&b, 6);

push(&b, 4);

push(&b, 2);

intersect = sortedIntersect(a, b);

cout<<"\n Linked list containing common items of a & b \n ";

printList(intersect);

}

**Output:**

**Program 66 : Delete alternate nodes of a Linked List.**

#include<iostream>

using namespace std;

struct Node

{

int data;

struct Node \*next;

};

void deleteAlt(struct Node \*head)

{

if (head == NULL)

return;

struct Node \*prev = head;

struct Node \*Node = head->next;

while (prev != NULL && Node != NULL)

{

prev->next = Node->next;

delete(Node);

prev = prev->next;

if (prev != NULL)

Node = prev->next;

}

}

void push(struct Node\*\* head\_ref, int new\_data)

{

struct Node\* new\_Node = new Node;

new\_Node->data = new\_data;

new\_Node->next = (\*head\_ref);

(\*head\_ref) = new\_Node;

}

void printList(struct Node \*Node)

{

while (Node != NULL)

{

cout<<Node->data;

Node = Node->next;

}

}

int main()

{

struct Node\* head = NULL;

push(&head, 5);

push(&head, 4);

push(&head, 3);

push(&head, 2);

push(&head, 1);

cout<<"\nList before calling deleteAlt() \n";

printList(head);

deleteAlt(head);

cout<<"\nList after calling deleteAlt() \n";

printList(head);

}

**Output:**

**Program 67 : Quick Sort with double linked list**

#include <iostream>

using namespace std;

struct Node

{

int data;

struct Node \*next;

struct Node \*prev;

};

void swap ( int\* a, int\* b )

{ int t = \*a; \*a = \*b; \*b = t; }

struct Node \*lastNode(Node \*root)

{

while (root && root->next)

root = root->next;

return root;

}

Node\* partition(Node \*l, Node \*h)

{

int x = h->data;

Node \*i = l->prev;

for (Node \*j = l; j != h; j = j->next)

{

if (j->data <= x)

{

i = (i == NULL)? l : i->next;

swap(&(i->data), &(j->data));

}

}

i = (i == NULL)? l : i->next;

swap(&(i->data), &(h->data));

return i;

}

void \_quickSort(struct Node\* l, struct Node \*h)

{

if (h != NULL && l != h && l != h->next)

{

struct Node \*p = partition(l, h);

\_quickSort(l, p->prev);

\_quickSort(p->next, h);

}

}

void quickSort(struct Node \*head)

{

struct Node \*h = lastNode(head);

\_quickSort(head, h);

}

void printList(struct Node \*head)

{

while (head)

{

cout << head->data << " ";

head = head->next;

}

cout << endl;

}

void push(struct Node\*\* head\_ref, int new\_data)

{

struct Node\* new\_node = new Node;

new\_node->data = new\_data;

new\_node->prev = NULL;

new\_node->next = (\*head\_ref);

if ((\*head\_ref) != NULL) (\*head\_ref)->prev = new\_node ;

(\*head\_ref) = new\_node;

}

int main()

{

struct Node \*a = NULL;

push(&a, 5);

push(&a, 20);

push(&a, 4);

push(&a, 3);

push(&a, 30);

cout << "Linked List before sorting \n";

printList(a);

quickSort(a);

cout << "Linked List after sorting \n";

printList(a);

return 0;

}

**Output:**

**Program 68 : Merge Sort with linked list**

#include<iostream>

using namespace std;

struct Node

{

int data;

struct Node\* next;

};

struct Node\* SortedMerge(struct Node\* a, struct Node\* b);

void FrontBackSplit(struct Node\* source,

struct Node\*\* frontRef, struct Node\*\* backRef);

void MergeSort(struct Node\*\* headRef)

{

struct Node\* head = \*headRef;

struct Node\* a;

struct Node\* b;

if ((head == NULL) || (head->next == NULL))

{

return;

}

FrontBackSplit(head, &a, &b);

MergeSort(&a);

MergeSort(&b);

\*headRef = SortedMerge(a, b);

}

struct Node\* SortedMerge(struct Node\* a, struct Node\* b)

{

struct Node\* result = NULL;

if (a == NULL)

return(b);

else if (b==NULL)

return(a);

if (a->data <= b->data)

{

result = a;

result->next = SortedMerge(a->next, b);

}

else

{

result = b;

result->next = SortedMerge(a, b->next);

}

return(result);

}

void FrontBackSplit(struct Node\* source,

struct Node\*\* frontRef, struct Node\*\* backRef)

{

struct Node\* fast;

struct Node\* slow;

slow = source;

fast = source->next;

while (fast != NULL)

{

fast = fast->next;

if (fast != NULL)

{

slow = slow->next;

fast = fast->next;

}

}

\*frontRef = source;

\*backRef = slow->next;

slow->next = NULL;

}

void printList(struct Node \*node)

{

while(node!=NULL)

{

printf("%d ", node->data);

node = node->next;

}

}

void push(struct Node\*\* head\_ref, int new\_data)

{

struct Node\* new\_node = new Node;

new\_node->data = new\_data;

new\_node->next = (\*head\_ref);

(\*head\_ref) = new\_node;

}

int main()

{

struct Node\* res = NULL;

struct Node\* a = NULL;

push(&a, 15);

push(&a, 10);

push(&a, 5);

push(&a, 20);

push(&a, 3);

push(&a, 2);

MergeSort(&a);

printf("Sorted Linked List is: \n");

printList(a);

getchar();

return 0;

}

**Ouptut:**

**Program 69 :Insertion Sort with linked list**

#include<iostream>

using namespace std;

struct Node

{

int data;

struct Node\* next;

};

void sortedInsert(struct Node\*\*, struct Node\*);

void insertionSort(struct Node \*\*head\_ref)

{

struct Node \*sorted = NULL;

struct Node \*current = \*head\_ref;

while (current != NULL)

{

struct Node \*next = current->next;

sortedInsert(&sorted, current);

current = next;

}

\*head\_ref = sorted;

}

void sortedInsert(struct Node\*\* head\_ref, struct Node\* new\_node)

{

struct Node\* current;

if (\*head\_ref == NULL || (\*head\_ref)->data >= new\_node->data)

{

new\_node->next = \*head\_ref;

\*head\_ref = new\_node;

}

else

{

current = \*head\_ref;

while (current->next!=NULL &&

current->next->data < new\_node->data)

{

current = current->next;

}

new\_node->next = current->next;

current->next = new\_node;

}

}

void printList(struct Node \*head)

{

struct Node \*temp = head;

while(temp != NULL)

{

cout<<temp->data;

temp = temp->next;

}

}

void push(struct Node\*\* head\_ref, int new\_data)

{

struct Node\* new\_node = new Node;

new\_node->data = new\_data;

new\_node->next = (\*head\_ref);

(\*head\_ref) = new\_node;

}

int main()

{

struct Node \*a = NULL;

push(&a, 5);

push(&a, 20);

push(&a, 4);

push(&a, 3);

push(&a, 30);

cout<<"Linked List before sorting"<<endl;

printList(a);

insertionSort(&a);

cout<<"\nLinked List after sorting"<<endl;;

printList(a);

return 0;

}

**Output:**

**Program 70 : Bubble Sort with Linked List**

#include<iostream>

using namespace std;

struct Node

{

int data;

struct Node \*next;

};

void insertAtTheBegin(struct Node \*\*start\_ref, int data);

void bubbleSort(struct Node \*start);

void swap(struct Node \*a, struct Node \*b);

void printList(struct Node \*start);

int main()

{

int arr[] = {12, 56, 2, 11, 1, 90};

int list\_size, i;

struct Node \*start = NULL;

for (i = 0; i< 6; i++)

insertAtTheBegin(&start, arr[i]);

cout<<"\n Linked list before sorting ";

printList(start);

bubbleSort(start);

cout<<"\n Linked list after sorting ";

printList(start);

getchar();

return 0;

}

void insertAtTheBegin(struct Node \*\*start\_ref, int data)

{

struct Node \*ptr1 = new Node;

ptr1->data = data;

ptr1->next = \*start\_ref;

\*start\_ref = ptr1;

}

void printList(struct Node \*start)

{

struct Node \*temp = start;

cout<<endl;

while (temp!=NULL)

{

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

temp = temp->next;

}

}

void bubbleSort(struct Node \*start)

{

int swapped, i;

struct Node \*ptr1;

struct Node \*lptr = NULL;

if (start == NULL)

return;

do

{

swapped = 0;

ptr1 = start;

while (ptr1->next != lptr)

{

if (ptr1->data > ptr1->next->data)

{

swap(ptr1, ptr1->next);

swapped = 1;

}

ptr1 = ptr1->next;

}

lptr = ptr1;

}

while (swapped);

}

void swap(struct Node \*a, struct Node \*b)

{

int temp = a->data;

a->data = b->data;

b->data = temp;

}

Output: