C++ PROGRAMMING TASKS

1. Friendbinaryopr

2. Binaryoperatoroverloading

3. Functionoverloading

4. Linkedlist

5. Double Linked list

6. Circular Linked list

7. Localclass declaration

8. Purevirtualfunction

9. operatoroverloading

10. stack

11. sorting Using array

12. UsingNewOperatorOverloading

13. DoubleStorageMapUsingCopyConstructor

14. STL Library

15. DoubleStorageMapUsingCopyConstructor.

16. StorageListControl

17. Matrix.

1. Friend Binary Operator

#include <iostream>

using namespace std;

class Point {

int x, y;

public:

Point() {} // needed to construct temporaries

Point(int px, int py) {

x = px;

y = py;

}

void show() {

cout << x << " ";

cout << y << "\n";

}

friend Point operator+(Point op1, Point op2); // now a friend

Point operator-(Point op2);

Point operator=(Point op2);

Point operator++();

};

// Now, + is overloaded using friend function.

Point operator+(Point op1, Point op2)

{

Point temp;

temp.x = op1.x + op2.x;

temp.y = op1.y + op2.y;

return temp;

}

// Overload - for Point.

Point Point::operator-(Point op2)

{

Point temp;

// notice order of operands

temp.x = x - op2.x;

temp.y = y - op2.y;

return temp;

}

// Overload assignment for Point.

Point Point::operator=(Point op2)

{

x = op2.x;

y = op2.y;

return \*this; // i.e., return object that generated call

}

// Overload ++ for Point.

Point Point::operator++()

{

x++;

y++;

return \*this;

}

int main()

{

Point ob1(10, 20), ob2(5, 30);

ob1 = ob1 + ob2;

ob1.show();

return 0;

}

1. Binary Operator Overloading

2.1 Binary Operator Overloading

#include<iostream>

#include<conio.h>

//Standard namespace declaration

using namespace std;

class overloading

{

int value;

public:

void setValue(int temp)

{

value = temp;

}

overloading operator+(overloading ob)

{

overloading t;

t.value = value + ob.value;

return(t);

}

void display()

{

cout << value << endl;

}

};

//Main Functions

int main()

{

overloading obj1, obj2, result;

int a, b;

cout << "Enter the value of Complex Numbers a,b:";

cin >> a >> b;

obj1.setValue(a);

obj2.setValue(b);

result = obj1 + obj2;

cout << "Input Values:\n";

obj1.display();

obj2.display();

cout << "Result:";

result.display();

return 0;

}

2.2 Unary Operator Overloading

#include<iostream.h>

#include<conio.h>

class complex

{

int a,b,c;

public:

complex(){}

void getvalue()

{

cout<<"Enter the Two Numbers:";

cin>>a>>b;

}

void operator++()

{

a=++a;

b=++b;

}

void operator--()

{

a=--a;

b=--b;

}

void display()

{

cout<<a<<"+\t"<<b<<"i"<<endl;

}

};

void main()

{

//clrscr();

complex obj;

obj.getvalue();

obj++;

cout<<"Increment Complex Number\n";

obj.display();

obj--;

cout<<"Decrement Complex Number\n";

obj.display();

//getch();

}

1. Function Overloading

#include <iostream>

#include<stdlib.h>

#include<conio.h>

using namespace std;

#define pi 3.14

class fn

{

public:

void area(int); //circle

void area(int, int); //rectangle

void area(float, int, int); //triangle

};

void fn::area(int a)

{

cout << "Area of Circle: \n" << pi\*a\*a;

}

void fn::area(int a, int b)

{

cout << "Area of rectangle: \n" << a\*b;

}

void fn::area(float t, int a, int b)

{

cout << "Area of triangle: \n" << t\*a\*b;

}

int main()

{

int ch;

int a, b, r;

fn obj;

cout << "\n\t\tFunction Overloading";

cout << "\n 1.Area of Circle \n 2.Area of Rectangle \n3.Area of Triangle\n4.Exit\n:" ;

cout << "Enter your Choice : ";

cin >> ch;

switch (ch)

{

case 1:

cout << "Enter Radious of the Circle:";

cin >> r;

obj.area(r);

break;

case 2:

cout << "Enter Sides of the Rectangle:";

cin >> a >> b;

obj.area(a, b);

break;

case 3:

cout << "Enter Sides of the Triangle:";

cin >> a >> b;

obj.area(0.5, a, b);

break;

case 4:

exit(0);

}

return 0;

}

1. Linked List

#include<iostream>

#include<cstdio>

#include<cstdlib>

using namespace std;

/\*

\* Node Declaration

\*/

struct node

{

int info;

struct node \*next;

}\*start;

/\*

\* Class Declaration

\*/

class single\_llist

{

public:

node\* create\_node(int);

void insert\_begin();

void insert\_pos();

void insert\_last();

void delete\_pos();

void sort();

void search();

void update();

void reverse();

void display();

single\_llist()

{

start = NULL;

}

};

/\*

\* Main :contains menu

\*/

int main()

{

int choice, nodes, element, position, i;

single\_llist sl;

start = NULL;

while (1)

{

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

cout << endl << "Operations on singly linked list" << endl;

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

cout << "1.Insert Node at beginning" << endl;

cout << "2.Insert node at last" << endl;

cout << "3.Insert node at position" << endl;

cout << "4.Sort Link List" << endl;

cout << "5.Delete a Particular Node" << endl;

cout << "6.Update Node Value" << endl;

cout << "7.Search Element" << endl;

cout << "8.Display Linked List" << endl;

cout << "9.Reverse Linked List " << endl;

cout << "10.Exit " << endl;

cout << "Enter your choice : ";

cin >> choice;

switch (choice)

{

case 1:

cout << "Inserting Node at Beginning: " << endl;

sl.insert\_begin();

cout << endl;

break;

case 2:

cout << "Inserting Node at Last: " << endl;

sl.insert\_last();

cout << endl;

break;

case 3:

cout << "Inserting Node at a given position:" << endl;

sl.insert\_pos();

cout << endl;

break;

case 4:

cout << "Sort Link List: " << endl;

sl.sort();

cout << endl;

break;

case 5:

cout << "Delete a particular node: " << endl;

sl.delete\_pos();

break;

case 6:

cout << "Update Node Value:" << endl;

sl.update();

cout << endl;

break;

case 7:

cout << "Search element in Link List: " << endl;

sl.search();

cout << endl;

break;

case 8:

cout << "Display elements of link list" << endl;

sl.display();

cout << endl;

break;

case 9:

cout << "Reverse elements of Link List" << endl;

sl.reverse();

cout << endl;

break;

case 10:

cout << "Exiting..." << endl;

exit(1);

break;

default:

cout << "Wrong choice" << endl;

}

}

}

/\*

\* Creating Node

\*/

node \*single\_llist::create\_node(int value)

{

struct node \*temp, \*s;

temp = new(struct node);

if (temp == NULL)

{

cout << "Memory not allocated " << endl;

return 0;

}

else

{

temp->info = value;

temp->next = NULL;

return temp;

}

}

/\*

\* Inserting element in beginning

\*/

void single\_llist::insert\_begin()

{

int value;

cout << "Enter the value to be inserted: ";

cin >> value;

struct node \*temp, \*p;

temp = create\_node(value);

if (start == NULL)

{

start = temp;

start->next = NULL;

}

else

{

p = start;

start = temp;

start->next = p;

}

cout << "Element Inserted at beginning" << endl;

}

/\*

\* Inserting Node at last

\*/

void single\_llist::insert\_last()

{

int value;

cout << "Enter the value to be inserted: ";

cin >> value;

struct node \*temp, \*s;

temp = create\_node(value);

s = start;

while (s->next != NULL)

{

s = s->next;

}

temp->next = NULL;

s->next = temp;

cout << "Element Inserted at last" << endl;

}

/\*

\* Insertion of node at a given position

\*/

void single\_llist::insert\_pos()

{

int value, pos, counter = 0;

struct node \*temp, \*s, \*ptr= NULL;

cout << "Enter the value to be inserted: ";

cin >> value;

temp = create\_node(value);

cout << "Enter the postion at which node to be inserted: ";

cin >> pos;

int i;

s = start;

while (s != NULL)

{

s = s->next;

counter++;

}

if (pos == 1)

{

if (start == NULL)

{

start = temp;

start->next = NULL;

}

else

{

ptr = start;

start = temp;

start->next = ptr;

}

}

else if (pos > 1 && pos <= counter)

{

s = start;

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

{

ptr = s;

s = s->next;

}

ptr -> next = temp;

temp -> next = s;

}

else

{

cout << "Positon out of range" << endl;

}

}

/\*

\* Sorting Link List

\*/

void single\_llist::sort()

{

struct node \*ptr, \*s;

int value;

if (start == NULL)

{

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

return;

}

ptr = start;

while (ptr != NULL)

{

for (s = ptr->next; s != NULL; s = s->next)

{

if (ptr->info > s->info)

{

value = ptr->info;

ptr->info = s->info;

s->info = value;

}

}

ptr = ptr->next;

}

}

/\*

\* Delete element at a given position

\*/

void single\_llist::delete\_pos()

{

int pos, i, counter = 0;

if (start == NULL)

{

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

return;

}

cout << "Enter the position of value to be deleted: ";

cin >> pos;

struct node \*s, \*ptr= NULL;

s = start;

if (pos == 1)

{

start = s->next;

}

else

{

while (s != NULL)

{

s = s->next;

counter++;

}

if (pos > 0 && pos <= counter)

{

s = start;

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

{

ptr = s;

s = s->next;

}

ptr->next = s->next;

}

else

{

cout << "Position out of range" << endl;

}

free(s);

cout << "Element Deleted" << endl;

}

}

/\*

\* Update a given Node

\*/

void single\_llist::update()

{

int value, pos, i;

if (start == NULL)

{

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

return;

}

cout << "Enter the node postion to be updated: ";

cin >> pos;

cout << "Enter the new value: ";

cin >> value;

struct node \*s, \*ptr;

s = start;

if (pos == 1)

{

start->info = value;

}

else

{

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

{

if (s == NULL)

{

cout << "There are less than " << pos << " elements";

return;

}

s = s->next;

}

s->info = value;

}

cout << "Node Updated" << endl;

}

/\*

\* Searching an element

\*/

void single\_llist::search()

{

int value, pos = 0;

bool flag = false;

if (start == NULL)

{

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

return;

}

cout << "Enter the value to be searched: ";

cin >> value;

struct node \*s;

s = start;

while (s != NULL)

{

pos++;

if (s->info == value)

{

flag = true;

cout << "Element " << value << " is found at position " << pos << endl;

}

s = s->next;

}

if (!flag)

cout << "Element " << value << " not found in the list" << endl;

}

/\*

\* Reverse Link List

\*/

void single\_llist::reverse()

{

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

if (start == NULL)

{

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

return;

}

if (start->next == NULL)

{

return;

}

ptr1 = start;

ptr2 = ptr1->next;

ptr3 = ptr2->next;

ptr1->next = NULL;

ptr2->next = ptr1;

while (ptr3 != NULL)

{

ptr1 = ptr2;

ptr2 = ptr3;

ptr3 = ptr3->next;

ptr2->next = ptr1;

}

start = ptr2;

}

/\*

\* Display Elements of a link list

\*/

void single\_llist::display()

{

struct node \*temp;

if (start == NULL)

{

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

return;

}

temp = start;

cout << "Elements of list are: " << endl;

while (temp != NULL)

{

cout << temp->info << "->";

temp = temp->next;

}

cout << "NULL" << endl;

}

1. Doubly Linked List

#include<iostream>

#include<cstdio>

#include<cstdlib>

/\*

\* Node Declaration

\*/

using namespace std;

struct node

{

int info;

struct node \*next;

struct node \*prev;

}\*start;

/\*

Class Declaration

\*/

class double\_llist

{

public:

void create\_list(int value);

void add\_begin(int value);

void add\_after(int value, int position);

void delete\_element(int value);

void search\_element(int value);

void display\_dlist();

void count();

void reverse();

double\_llist()

{

start = NULL;

}

};

/\*

\* Main: Conatins Menu

\*/

int main()

{

int choice, element, position;

double\_llist dl;

while (1)

{

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

cout << endl << "Operations on Doubly linked list" << endl;

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

cout << "1.Create Node" << endl;

cout << "2.Add at begining" << endl;

cout << "3.Add after position" << endl;

cout << "4.Delete" << endl;

cout << "5.Display" << endl;

cout << "6.Count" << endl;

cout << "7.Reverse" << endl;

cout << "8.Quit" << endl;

cout << "Enter your choice : ";

cin >> choice;

switch (choice)

{

case 1:

cout << "Enter the element: ";

cin >> element;

dl.create\_list(element);

cout << endl;

break;

case 2:

cout << "Enter the element: ";

cin >> element;

dl.add\_begin(element);

cout << endl;

break;

case 3:

cout << "Enter the element: ";

cin >> element;

cout << "Insert Element after postion: ";

cin >> position;

dl.add\_after(element, position);

cout << endl;

break;

case 4:

if (start == NULL)

{

cout << "List empty,nothing to delete" << endl;

break;

}

cout << "Enter the element for deletion: ";

cin >> element;

dl.delete\_element(element);

cout << endl;

break;

case 5:

dl.display\_dlist();

cout << endl;

break;

case 6:

dl.count();

break;

case 7:

if (start == NULL)

{

cout << "List empty,nothing to reverse" << endl;

break;

}

dl.reverse();

cout << endl;

break;

case 8:

exit(1);

default:

cout << "Wrong choice" << endl;

}

}

return 0;

}

/\*

\* Create Double Link List

\*/

void double\_llist::create\_list(int value)

{

struct node \*s, \*temp;

temp = new(struct node);

temp->info = value;

temp->next = NULL;

if (start == NULL)

{

temp->prev = NULL;

start = temp;

}

else

{

s = start;

while (s->next != NULL)

s = s->next;

s->next = temp;

temp->prev = s;

}

}

/\*

\* Insertion at the beginning

\*/

void double\_llist::add\_begin(int value)

{

if (start == NULL)

{

cout << "First Create the list." << endl;

return;

}

struct node \*temp;

temp = new(struct node);

temp->prev = NULL;

temp->info = value;

temp->next = start;

start->prev = temp;

start = temp;

cout << "Element Inserted" << endl;

}

/\*

\* Insertion of element at a particular position

\*/

void double\_llist::add\_after(int value, int pos)

{

if (start == NULL)

{

cout << "First Create the list." << endl;

return;

}

struct node \*tmp, \*q;

int i;

q = start;

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

{

q = q->next;

if (q == NULL)

{

cout << "There are less than ";

cout << pos << " elements." << endl;

return;

}

}

tmp = new(struct node);

tmp->info = value;

if (q->next == NULL)

{

q->next = tmp;

tmp->next = NULL;

tmp->prev = q;

}

else

{

tmp->next = q->next;

tmp->next->prev = tmp;

q->next = tmp;

tmp->prev = q;

}

cout << "Element Inserted" << endl;

}

/\*

\* Deletion of element from the list

\*/

void double\_llist::delete\_element(int value)

{

struct node \*tmp, \*q;

/\*first element deletion\*/

if (start->info == value)

{

tmp = start;

start = start->next;

start->prev = NULL;

cout << "Element Deleted" << endl;

free(tmp);

return;

}

q = start;

while (q->next->next != NULL)

{

/\*Element deleted in between\*/

if (q->next->info == value)

{

tmp = q->next;

q->next = tmp->next;

tmp->next->prev = q;

cout << "Element Deleted" << endl;

free(tmp);

return;

}

q = q->next;

}

/\*last element deleted\*/

if (q->next->info == value)

{

tmp = q->next;

free(tmp);

q->next = NULL;

cout << "Element Deleted" << endl;

return;

}

cout << "Element " << value << " not found" << endl;

}

/\*

\* Display elements of Doubly Link List

\*/

void double\_llist::display\_dlist()

{

struct node \*q;

if (start == NULL)

{

cout << "List empty,nothing to display" << endl;

return;

}

q = start;

cout << "The Doubly Link List is :" << endl;

while (q != NULL)

{

cout << q->info << " <-> ";

q = q->next;

}

cout << "NULL" << endl;

}

/\*

\* Number of elements in Doubly Link List

\*/

void double\_llist::count()

{

struct node \*q = start;

int cnt = 0;

while (q != NULL)

{

q = q->next;

cnt++;

}

cout << "Number of elements are: " << cnt << endl;

}

/\*

\* Reverse Doubly Link List

\*/

void double\_llist::reverse()

{

struct node \*p1, \*p2;

p1 = start;

p2 = p1->next;

p1->next = NULL;

p1->prev = p2;

while (p2 != NULL)

{

p2->prev = p2->next;

p2->next = p1;

p1 = p2;

p2 = p2->prev;

}

start = p1;

cout << "List Reversed" << endl;

}

1. Circular Linked List

#include<iostream>

#include<cstdio>

#include<cstdlib>

using namespace std;

/\*

\* Node Declaration

\*/

struct node

{

int info;

struct node \*next;

}\*last;

/\*

\* Class Declaration

\*/

class circular\_llist

{

public:

void create\_node(int value);

void add\_begin(int value);

void add\_after(int value, int position);

void delete\_element(int value);

void search\_element(int value);

void display\_list();

void update();

void sort();

circular\_llist()

{

last = NULL;

}

};

/\*

\* Main :contains menu

\*/

int main()

{

int choice, element, position;

circular\_llist cl;

while (1)

{

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

cout << endl << "Circular singly linked list" << endl;

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

cout << "1.Create Node" << endl;

cout << "2.Add at beginning" << endl;

cout << "3.Add after" << endl;

cout << "4.Delete" << endl;

cout << "5.Search" << endl;

cout << "6.Display" << endl;

cout << "7.Update" << endl;

cout << "8.Sort" << endl;

cout << "9.Quit" << endl;

cout << "Enter your choice : ";

cin >> choice;

switch (choice)

{

case 1:

cout << "Enter the element: ";

cin >> element;

cl.create\_node(element);

cout << endl;

break;

case 2:

cout << "Enter the element: ";

cin >> element;

cl.add\_begin(element);

cout << endl;

break;

case 3:

cout << "Enter the element: ";

cin >> element;

cout << "Insert element after position: ";

cin >> position;

cl.add\_after(element, position);

cout << endl;

break;

case 4:

if (last == NULL)

{

cout << "List is empty, nothing to delete" << endl;

break;

}

cout << "Enter the element for deletion: ";

cin >> element;

cl.delete\_element(element);

cout << endl;

break;

case 5:

if (last == NULL)

{

cout << "List Empty!! Can't search" << endl;

break;

}

cout << "Enter the element to be searched: ";

cin >> element;

cl.search\_element(element);

cout << endl;

break;

case 6:

cl.display\_list();

break;

case 7:

cl.update();

break;

case 8:

cl.sort();

break;

case 9:

exit(1);

break;

default:

cout << "Wrong choice" << endl;

}

}

return 0;

}

/\*

\* Create Circular Link List

\*/

void circular\_llist::create\_node(int value)

{

struct node \*temp;

temp = new(struct node);

temp->info = value;

if (last == NULL)

{

last = temp;

temp->next = last;

}

else

{

temp->next = last->next;

last->next = temp;

last = temp;

}

}

/\*

\* Insertion of element at beginning

\*/

void circular\_llist::add\_begin(int value)

{

if (last == NULL)

{

cout << "First Create the list." << endl;

return;

}

struct node \*temp;

temp = new(struct node);

temp->info = value;

temp->next = last->next;

last->next = temp;

}

/\*

\* Insertion of element at a particular place

\*/

void circular\_llist::add\_after(int value, int pos)

{

if (last == NULL)

{

cout << "First Create the list." << endl;

return;

}

struct node \*temp, \*s;

s = last->next;

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

{

s = s->next;

if (s == last->next)

{

cout << "There are less than ";

cout << pos << " in the list" << endl;

return;

}

}

temp = new(struct node);

temp->next = s->next;

temp->info = value;

s->next = temp;

/\*Element inserted at the end\*/

if (s == last)

{

last = temp;

}

}

/\*

\* Deletion of element from the list

\*/

void circular\_llist::delete\_element(int value)

{

struct node \*temp, \*s;

s = last->next;

/\* If List has only one element\*/

if (last->next == last && last->info == value)

{

temp = last;

last = NULL;

free(temp);

cout << "element deleted";

return;

}

if (s->info == value) /\*First Element Deletion\*/

{

temp = s;

last->next = s->next;

free(temp);

cout << "element deleted";

return;

}

while (s->next != last)

{

/\*Deletion of Element in between\*/

if (s->next->info == value)

{

temp = s->next;

s->next = temp->next;

free(temp);

cout << "Element " << value;

cout << " deleted from the list" << endl;

return;

}

s = s->next;

}

/\*Deletion of last element\*/

if (s->next->info == value)

{

temp = s->next;

s->next = last->next;

free(temp);

last = s;

return;

}

cout << "Element " << value << " not found in the list" << endl;

}

/\*

\* Search element in the list

\*/

void circular\_llist::search\_element(int value)

{

struct node \*s;

int counter = 0;

s = last->next;

while (s != last)

{

counter++;

if (s->info == value)

{

cout << "Element " << value;

cout << " found at position " << counter << endl;

return;

}

s = s->next;

}

if (s->info == value)

{

counter++;

cout << "Element " << value;

cout << " found at position " << counter << endl;

return;

}

cout << "Element " << value << " not found in the list" << endl;

}

/\*

\* Display Circular Link List

\*/

void circular\_llist::display\_list()

{

struct node \*s;

if (last == NULL)

{

cout << "List is empty, nothing to display" << endl;

return;

}

s = last->next;

cout << "Circular Link List: " << endl;

while (s != last)

{

cout << s->info << "->";

s = s->next;

}

cout << s->info << endl;

}

/\*

\* Update Circular Link List

\*/

void circular\_llist::update()

{

int value, pos, i;

if (last == NULL)

{

cout << "List is empty, nothing to update" << endl;

return;

}

cout << "Enter the node position to be updated: ";

cin >> pos;

cout << "Enter the new value: ";

cin >> value;

struct node \*s;

s = last->next;

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

{

if (s == last)

{

cout << "There are less than " << pos << " elements.";

cout << endl;

return;

}

s = s->next;

}

s->info = value;

cout << "Node Updated" << endl;

}

/\*

\* Sort Circular Link List

\*/

void circular\_llist::sort()

{

struct node \*s, \*ptr;

int temp;

if (last == NULL)

{

cout << "List is empty, nothing to sort" << endl;

return;

}

s = last->next;

while (s != last)

{

ptr = s->next;

while (ptr != last->next)

{

if (ptr != last->next)

{

if (s->info > ptr->info)

{

temp = s->info;

s->info = ptr->info;

ptr->info = temp;

}

}

else

{

break;

}

ptr = ptr->next;

}

s = s->next;

}

}

1. Local Class Declartion

#include <iostream>

using namespace std;

int main()

{

class loc

{

int regno;

float fee;

public:

void readdata()

{

cout << "enter the reg no and fee";

cin >> regno >> fee;

}

void writedata()

{

cout << "the regno you have enterd is" << regno;

cout << "the fees you have netered is" << fee;

}

};

loc lt;

lt.readdata();

lt.writedata();

getchar();

cin.get();

return 0;

}

1. Pure Virtual Function

#include <iostream>

using namespace std;

class base

{

private: int x;

float y;

public: virtual void getdata();

virtual void display();

};

class dev : public base

{

private: int roll;

char name[20];

public: void getdata();

void display();

};

void base::getdata() { }

void base::display() { }

void dev::getdata()

{

cout << " Enter Roll of the Student ";

cin >> roll;

cout << "Enter name of the student";

cin >> name;

}

void dev::display()

{

cout << "Name is : " << name << endl;

cout << " Roll no is : " << roll << endl;

}

void main()

{

base \* ptr;

dev obj;

ptr = &obj;

ptr->getdata();

ptr->display();

}

1. Operator Overloading

#include <iostream>

using namespace std;

class example

{

public:

int a;

int b;

example operator+(const example& obj);

void operator=(const example& obj);

};

void example::operator=(const example& obj)

{

(\*this).a = obj.a;

(\*this).b = obj.b;

return;

}

example example::operator+(const example& obj2)

{

example tmp\_obj = \*this;

tmp\_obj.a = tmp\_obj.a + obj2.a;

tmp\_obj.b = tmp\_obj.b + obj2.b;

return tmp\_obj;

}

int main(void)

{

example obj1, obj2, obj3;

obj1.a = 1;

obj1.b = 1;

obj2.a = 2;

obj2.b = 2;

obj3.a = 0;

obj3.b = 0;

obj3 = obj1 + obj2;

/\*std::cout << obj3.a << " " << obj3.b << "\n";

\*/

cout << obj3.a << " " << obj3.b << "\n";

return 0;

}

1. Stack

#include<iostream>

#include<cstdlib>

using namespace std;

/\*

\* Node Declaration

\*/

struct node

{

int info;

struct node \*link;

}\*top;

/\*

\* Class Declaration

\*/

class stack\_list

{

public:

node \*push(node \*, int);

node \*pop(node \*);

void traverse(node \*);

stack\_list()

{

top = NULL;

}

};

/\*

\* Main: Contains Menu

\*/

int main()

{

int choice, item;

stack\_list sl;

while (1)

{

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

cout << "Operations on Stack" << endl;

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

cout << "1.Push Element into the Stack" << endl;

cout << "2.Pop Element from the Stack" << endl;

cout << "3.Traverse the Stack" << endl;

cout << "4.Quit" << endl;

cout << "Enter your Choice: ";

cin >> choice;

switch (choice)

{

case 1:

cout << "Enter value to be pushed into the stack: ";

cin >> item;

top = sl.push(top, item);

break;

case 2:

top = sl.pop(top);

break;

case 3:

sl.traverse(top);

break;

case 4:

exit(1);

break;

default:

cout << "Wrong Choice" << endl;

}

}

return 0;

}

/\*

\* Push Element into the Stack

\*/

node \*stack\_list::push(node \*top, int item)

{

node \*tmp;

tmp = new (struct node);

tmp->info = item;

tmp->link = top;

top = tmp;

return top;

}

/\*

\* Pop Element from the Stack

\*/

node \*stack\_list::pop(node \*top)

{

node \*tmp;

if (top == NULL)

cout << "Stack is Empty" << endl;

else

{

tmp = top;

cout << "Element Popped: " << tmp->info << endl;

top = top->link;

free(tmp);

}

return top;

}

/\*

\* Traversing the Stack

\*/

void stack\_list::traverse(node \*top)

{

node \*ptr;

ptr = top;

if (top == NULL)

cout << "Stack is empty" << endl;

else

{

cout << "Stack elements :" << endl;

while (ptr != NULL)

{

cout << ptr->info << endl;

ptr = ptr->link;

}

}

}

1. Sorting Using Array

11.1 Ascending Order

#include <iostream>

#include <conio.h>

using namespace std;

#define MAXSIZE 10

int main()

{

int array[MAXSIZE];

int i, j, num, temp;

cout<<"Enter the value of num \n";

cin >> num;

cout << "Enter the elements one by one \n";

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

{

cin>> array[i];

}

cout << "Input array is \n";

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

{

cout << array[i] << "," ;

}

/\* Bubble sorting begins \*/

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

{

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

{

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

{

temp = array[j];

array[j] = array[j + 1];

array[j + 1] = temp;

}

}

}

cout << "\n";

cout << "Sorted array is...\n";

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

{

cout << array[i] << ",";

}

return 0;

}

11.2 Descending Order

#include <iostream>

#include<conio.h>

using namespace std;

int main()

{

int number[30];

int i, j, a, n;

cout<<"Enter the value of N\n";

cin>>n;

cout << "Enter the numbers \n";

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

cin>>number[i];

/\* sorting begins ... \*/

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

{

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

{

if (number[i] < number[j])

{

a = number[i];

number[i] = number[j];

number[j] = a;

}

}

}

cout << "The numbers arranged in descending order are given below\n";

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

{

cout << number[i] << ",";

}

return 0;

}

1. Using new operator Overloading

#include <iostream>

#include <stdlib.h>

using namespace std;

using namespace std;

void\* operator new(size\_t n)

{

cout << endl << "overloaded new";

void \*ptr;

ptr = malloc(n);

return ptr;

}

void operator delete(void \*p)

{

cout << endl << "overloaded delete";

free(p);

}

int main()

{

int \*p = new int;

\*p = 20;

cout << "\n The value is : " << \*p; //value of 20

delete p;

cout << "\n The value after deleting is : " << p; //some invalid value

return 0;

}

1. Double storage map using Copy Constructor

#include <iostream>

#include <map>

#include <string>

#include <cstdlib>

using namespace std;

int main()

{

map<char, int> mp;

map<char, int>::iterator it;

int choice, item;

char s;

while (1)

{

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

cout << "Map Implementation in Stl" << endl;

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

cout << "1.Insert Element into the Map" << endl;

cout << "2.Delete Element of the Map" << endl;

cout << "3.Size of the Map" << endl;

cout << "4.Find Element at a key in Map" << endl;

cout << "5.Dislplay by Iterator" << endl;

cout << "6.Count Elements at a specific key" << endl;

cout << "7.Exit" << endl;

cout << "Enter your Choice: ";

cin >> choice;

switch (choice)

{

case 1:

cout << "Enter value to be inserted: ";

cin >> item;

cout << "Enter the key: ";

cin >> s;

mp.insert(pair<char, int>(s, item));

break;

case 2:

cout << "Enter the mapped string to be deleted: ";

cin >> s;

mp.erase(s);

break;

case 3:

cout << "Size of Map: ";

cout << mp.size() << endl;

break;

case 4:

cout << "Enter the key at which value to be found: ";

cin >> s;

if (mp.count(s) != 0)

cout << mp.find(s)->second << endl;

else

cout << "No Element Found" << endl;

break;

case 5:

cout << "Displaying Map by Iterator: ";

for (it = mp.begin(); it != mp.end(); it++)

{

cout << (\*it).first << ": " << (\*it).second << endl;

}

break;

case 6:

cout << "Enter the key at which number of values to be counted: ";

cin >> s;

cout << mp.count(s) << endl;

break;

case 7:

exit(1);

break;

default:

cout << "Wrong Choice" << endl;

}

}

return 0;

}

1. STL Library

14.1) Array STL

#include <iostream>

#include <array>

#include <string>

#include <cstdlib>

using namespace std;

int main()

{

array<int, 5> arr;

array<int, 5>::iterator it;

int choice, item;

arr.fill(0);

int count = 0;

while (1)

{

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

cout << "Array Implementation in Stl" << endl;

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

cout << "1.Insert Element into the Array" << endl;

cout << "2.Size of the array" << endl;

cout << "3.Front Element of Array" << endl;

cout << "4.Back Element of Array" << endl;

cout << "5.Display elements of the Array" << endl;

cout << "6.Exit" << endl;

cout << "Enter your Choice: ";

cin >> choice;

switch (choice)

{

case 1:

cout << "Enter value to be inserted: ";

cin >> item;

arr.at(count) = item;

count++;

break;

case 2:

cout << "Size of the Array: ";

cout << arr.size() << endl;

break;

case 3:

cout << "Front Element of the Array: ";

cout << arr.front() << endl;

break;

case 4:

cout << "Back Element of the array: ";

cout << arr.back() << endl;

break;

case 5:

for (it = arr.begin(); it != arr.end(); ++it)

cout << " " << \*it;

cout << endl;

break;

case 6:

exit(1);

break;

default:

cout << "Wrong Choice" << endl;

}

}

return 0;

}

14.2) Container STL

#include <iostream>

#include <list>

#include <cstdlib>

using namespace std;

int main()

{

int choice, item;

list<int> lt;

list<int>::iterator it;

while (1)

{

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

cout << "Sorting Containers Implementation in Stl" << endl;

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

cout << "1.Insert Element into the List" << endl;

cout << "2.Display Sorted Elements" << endl;

cout << "3.Exit" << endl;

cout << "Enter your Choice: ";

cin >> choice;

switch (choice)

{

case 1:

cout << "Enter the element to be inserted: ";

cin >> item;

lt.push\_back(item);

break;

case 2:

lt.sort();

cout << "Elements of Sorted List: ";

for (it = lt.begin(); it != lt.end(); ++it)

cout << " " << \*it;

cout << endl;

break;

case 3:

exit(1);

break;

default:

cout << "Wrong Choice" << endl;

}

}

return 0;

}

14.3) QUEUE STL

#include <iostream>

#include <queue>

#include <string>

#include <cstdlib>

using namespace std;

int main()

{

queue<int> q;

int choice, item;

while (1)

{

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

cout << "Queue Implementation in Stl" << endl;

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

cout << "1.Insert Element into the Queue" << endl;

cout << "2.Delete Element from the Queue" << endl;

cout << "3.Size of the Queue" << endl;

cout << "4.Front Element of the Queue" << endl;

cout << "5.Last Element of the Queue" << endl;

cout << "6.Exit" << endl;

cout << "Enter your Choice: ";

cin >> choice;

switch (choice)

{

case 1:

cout << "Enter value to be inserted: ";

cin >> item;

q.push(item);

break;

case 2:

item = q.front();

q.pop();

cout << "Element " << item << " Deleted" << endl;

break;

case 3:

cout << "Size of the Queue: ";

cout << q.size() << endl;

break;

case 4:

cout << "Front Element of the Queue: ";

cout << q.front() << endl;

break;

case 5:

cout << "Back Element of the Queue: ";

cout << q.back() << endl;

break;

case 6:

exit(1);

break;

default:

cout << "Wrong Choice" << endl;

}

}

return 0;

}

14.4) String STL

#include <iostream>

#include <string>

#include <cstdlib>

using namespace std;

int main()

{

int choice, pos, len;

string::iterator it;

size\_t found;

string s, str = "This is a Test String.";

cout<<"Initial String is--> "<<str<<endl;

while (1)

{

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

cout<<"String Implementation in Stl"<<endl;

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

cout<<"1.Insert Substring in a String"<<endl;

cout<<"2.Erase Substring from a String"<<endl;

cout<<"3.Append Substring to a String"<<endl;

cout<<"4.Replace the String with a Substrng"<<endl;

cout<<"5.Size of the String"<<endl;

cout<<"6.Find substring in a String"<<endl;

cout<<"7.Display the String"<<endl;

cout<<"8.Exit"<<endl;

cout<<"Enter your Choice: ";

cin>>choice;

switch(choice)

{

case 1:

cout<<"Enter the substring to be inserted: ";

cin>>s;

cout<<"Position after which substring to be inserted: ";

cin>>pos;

if (pos <= str.length())

str.insert(pos, s);

else

cout<<"Position out of bounds"<<endl;

break;

case 2:

cout<<"Position after which substring to be erased: ";

cin>>pos;

cout<<"Length of the substring to be deleted: ";

cin>>len;

str.erase(pos, len);

break;

case 3:

s = " This is an appended string.";

str.append(s);

break;

case 4:

s = "nnn example";

str.replace(9, 5, s);

break;

case 5:

cout<<"Size of the string: "<<str.size()<<endl;

break;

case 6:

cout<<"Enter substring to be found: ";

cin>>s;

found = str.find(s);

if (found != string::npos)

cout <<"Substring "<<s<<" found at " << found <<endl;

else

cout <<"Substring "<<s<<" not found"<<endl;

break;

case 7:

for (it = str.begin(); it != str.end(); ++it)

cout<<\*it;

cout<<endl;

break;

case 8:

exit(1);

break;

default:

cout<<"Wrong Choice"<<endl;

}

}

return 0;

}

1. Storage List Control

15.1)

#include<iostream>

using namespace std;

int g; //global variable, initially holds 0

void test\_function()

{

static int s; //static variable, initially holds 0

register int r; //register variable

r = 5;

s = s + r \* 2;

cout << "Inside test\_function" << endl;

cout << "g = " << g << endl;

cout << "s = " << s << endl;

cout << "r = " << r << endl;

}

int main()

{

int a; //automatic variable

g = 25;

a = 17;

test\_function();

cout << "Inside main" << endl;

cout << "a = " << a << endl;

cout << "g = " << g << endl;

test\_function();

return 0;

}

15.2) For Mutable Class

#include<iostream>

using namespace std;

class test

{

mutable int a;

int b;

public:

test(int x, int y)

{

a = x;

b = y;

}

void square\_a() const

{

a = a\*a;

}

void display() const

{

cout << "a = " << a << endl;

cout << "b = " << b << endl;

}

};

int main()

{

const test x(2, 3);

cout << "Initial value" << endl;

x.display();

x.square\_a();

cout << "Final value" << endl;

x.display();

return 0;

}

1. Matrix

#include "iostream"

using namespace std;

# include<stdio.h>

void display(int[][3]);

void main()

{

int c;

void func1();

void func2();

void func3();

void func4();

void func5();

//clrscr();

cout<<"\n- : Matrix Manipulation Functions (for 3 X 3 Matrix) : -";

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

cout<<"\n Matrix Addition : 1";

cout<<"\n Matrix Subtraction : 2";

cout<<"\n Matrix Multiplication : 3";

cout<<"\n Find Transpose Matrix : 4";

cout<<"\n Matrix is Symmetric or not : 6";

cout<<"\n Enter Your Choice : ";

cin >>c;

switch (c)

{

case 1:

func1();

break;

case 2:

func2();

break;

case 3:

func3();

break;

case 4:

func4();

break;

case 5:

func5();

break;

default:

cout<<"\nInvalid Choice";

}

//getch();

}

void func1()

{

int x[3][3], y[3][3], z[3][3];

void getmatrix(int[][3]);

void addition(int[][3], int[][3], int[][3]);

//clrscr();

getmatrix(x);

getmatrix(y);

addition(x, y, z);

cout<<"\n - : Matrix 1: - \n";

display(x);

cout<<"\n - : Matrix 2: - \n";

display(y);

cout<<"\n - : Matrix Addition (Result): - \n";

display(z);

}

void getmatrix(int t[][3])

{

int i, j;

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

{

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

{

cout<<"Enter element ";

cin>>t[i][j];

}

}

}

void addition(int p[][3], int q[][3], int r[][3])

{

int i, j;

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

{

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

r[i][j] = p[i][j] + q[i][j];

}

}

void func2()

{

int x[3][3], y[3][3], z[3][3];

void getmatrix(int[][3]);

void subtraction(int[][3], int[][3], int[][3]);

//clrscr();

getmatrix(x);

getmatrix(y);

subtraction(x, y, z);

cout<<"\n - : Matrix 1: - \n";

display(x);

cout<<"\n - : Matrix 2: - \n";

display(y);

cout<<"\n - : Matrix Subtraction (Result): - \n";

display(z);

}

void subtraction(int p[3][3], int q[3][3], int r[3][3])

{

int i, j;

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

{

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

r[i][j] = p[i][j] - q[i][j];

}

}

void func3()

{

int x[3][3], y[3][3], z[3][3];

void getmatrix(int[][3]);

void multiplication(int[][3], int[][3], int[][3]);

//clrscr();

getmatrix(x);

getmatrix(y);

multiplication(x, y, z);

cout<<"\n - : Matrix 1: - \n";

display(x);

cout<<"\n - : Matrix 2: - \n";

display(y);

cout<<"\n - : Matrix Multiplication (Result): - \n";

display(z);

}

void multiplication(int p[][3], int q[3][3], int r[3][3])

{

int i, j, k;

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

//condition i< total row of matrix1

{

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

//condition i< total col of matrix1 or//condition i< total row of matrix2

{

r[i][j] = 0;

for (k = 0; k<3; k++) //condition i< total col of matrix2

r[i][j] = r[i][j] + (p[i][j] \* q[j][k]);

}

}

}

void func4()

{

int x[3][3], y[3][3];

void getmatrix(int[][3]);

void transpose(int[][3], int[][3]);

//clrscr();

getmatrix(x);

transpose(x, y);

cout<<"\n - : Matrix 1: - \n";

display(x);

cout<<"\n - : Transpose Matrix : - \n";

display(y);

}

void transpose(int p[][3], int q[][3])

{

int i, j;

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

{

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

q[i][j] = p[j][i];

}

}

void func5()

{

int x[3][3], y[3][3];

void getmatrix(int[][3]);

void transpose(int[][3], int[][3]);

int symmetric(int[][3], int[][3]);

//clrscr();

getmatrix(x);

transpose(x, y);

if (symmetric(x, y) == 1)

cout<<"\nMatrix is Symmetric";

else

cout<<"\nMatrix is Not Symmetric";

}

int symmetric(int p[][3], int q[][3])

{

int i, j;

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

{

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

{

if (q[i][j] != p[i][j])

return 0;

}

}

return 1;

}

void display(int m[][3])

{

int i, j;

cout<<"\n\n";

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

{

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

cout<<m[i][j]<< "\t";

cout<<"\n";

}

}