**Heaven’s Light is Our Guide**



**Rajshahi University of Engineering and Technology**

**Department of Computer Science and Engineering**

**Course No:** CSE.1204

**Course Title:** Sessional based on CSE.1203 (Object Oriented Programming)

**Lab Report No:** 03

**Lab Report On:** Operator Overloading,Stack & Queue with Class in C++

**Submitted By** **Submitted To**

Md. Ariful Islam Md. Asifur Rahman

Roll No: 1803046 Lecturer

Section: A Dept. of CSE,RUET

Department: CSE

**Problem No:** 01

**Problem Statement:** Implementation of Operator Overloading.

coord

|  |
| --- |
| int x;  int y;  int z; |
| coord ( );  coord (int,int,int);  void get (int &i, int &j,int &k);  coord operator + (coord obj); |

**Theory :**

Operator overloading is a compile-time polymorphism in which the operator is overloaded to provide the special meaning to the user-defined data type. Operator overloading is used to overload or redefines most of the operators available in C++. It is used to perform the operation on the user-defined data type.

The advantage of Operators overloading is to perform different operations on the same operand.

**Rules for Operator Overloading**

* Existing operators can only be overloaded, but the new operators cannot be overloaded.
* The overloaded operator contains atleast one operand of the user-defined data type.
* We cannot use friend function to overload certain operators. However, the member function can be used to overload those operators.
* When unary operators are overloaded through a member function take no explicit arguments, but, if they are overloaded by a friend function, takes one argument.
* When binary operators are overloaded through a member function takes one explicit argument, and if they are overloaded through a friend function takes two explicit arguments.

**Source Code :**

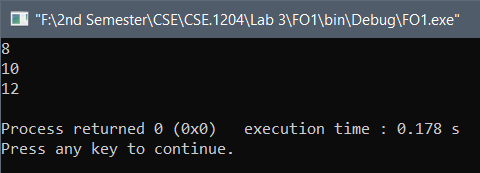
1. main.h

|  |
| --- |
| #include <iostream>  #include "coord.h"  using namespace std;  int main()  {  coord o1(3,4,5),o2(5,6,7),o3;  int x,y,z;  o3=o1+o2;  o3.get(x,y,z);  return 0;  } |

|  |
| --- |
| #ifndef COORD\_H  #define COORD\_H  class coord  {  int x;  int y;  int z;  public:  coord();  coord(int,int,int);  void get(int &i,int &j,int &k);  coord operator+(coord ob);  };  #endif // COORD\_H |

1. coord.h
2. coord.cpp

|  |
| --- |
| #include <iostream>  #include "coord.h"  using namespace std;  coord::coord()  {  x=0;  y=0;  z=0;  }  coord::coord(int a,int b,int c)  {  x=a;  y=b;  z=c;  }  void coord::get(int &i,int &j,int &k)  {  i=x;  cout<<i<<endl;  j=y;  cout<<j<<endl;  k=z;  cout<<k<<endl;  }  coord coord::operator+(coord ob)  {  coord temp;  temp.x=x+ob.x;  temp.y=y+ob.y;  temp.z=z+ob.z;  return temp;  } |

**Output :**

**Problem No:** 02

**Problem Statement:** Implementation of Stack.

stack

|  |
| --- |
| int i;  int ax[100]; |
| void push(int);  void pop();  void show(); |

**Theory :** Stacks are a type of container adaptors with LIFO(Last In First Out) type of working, where a new element is added at one end and (top) an element is removed from that end only.

We used three public functions in the class :

* 1. **push:** This function take data as input.
  2. **pop:** This function delete the last inputted data.
  3. **show:** this function shows all the remaining data.

**Source Code :**

1. main.h

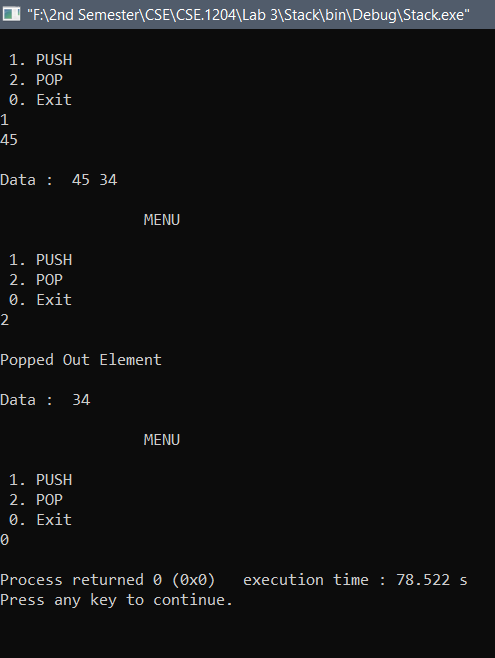
|  |
| --- |
| #include <iostream>  #include "stack.h"  using namespace std;  void menu()  {  cout<<"\n\t\t"<<"MENU"<<"\n"<<endl;  cout<<" 1. PUSH"<<"\n"<<" 2. POP";  cout<<"\n"<<" 0. Exit"<<endl;  }  int main()  {  stack o;  int a,i,j,k;  menu();  cin>>i;  while(i!=0)  {  if(i==1)  {  cin>>a;  o.push(a);  o.show();  menu();  cin>>i;  }  else if(i==2)  {  o.pop();  o.show();  menu();  cin>>i;  }  else if(i==0)  break;  else  {  cout<<"Wrong Input"<<"\n"<<endl;  menu();  cin>>i;  }  }  return 0;  } |

1. stack.h

|  |
| --- |
| #ifndef STACK\_H  #define STACK\_H  class stack  {  int i=0;  int ax[100]={-1};  public:  void push(int);  void pop();  void show();  };  #endif // STACK\_H |

1. stack.cpp

|  |
| --- |
| #include <iostream>  #include "stack.h"  using namespace std;  void stack::push(int a)  {  ax[i]=a;  i=i+1;  cout<<"\n\n"<<"Data : ";  }  void stack::pop()  {  if(i>0)  {  ax[i]=-1;  i=i-1;  cout<<"\n"<<"Popped Out Element"<<endl;  if(i!=0)  cout<<"\n"<<"Data : ";  }  else  cout<<"No Elements..."<<"\n"<<endl;  }  void stack::show()  {  int j;  for(j=i-1;j>=0;j--)  cout<<ax[j]<<" ";  cout<<endl;  } |

**Output:**

**Problem No:** 03

**Problem Statement:** Implementation of Queue.

queue

|  |
| --- |
| int i;  int ax[100]; |
| void push(int);  void pop();  void show(); |

**Theory :** Queues are a type of container adaptors which operate in a first in first out (FIFO) type of arrangement. Elements are inserted at the back (end) and are deleted from the front.

We used three public functions in the class :

1. **push:** This function take data as input.
2. **pop:** This function delete the **first** inputted data.
3. **show:** this function shows all the remaining data.

**Source Code :**

1. main.h

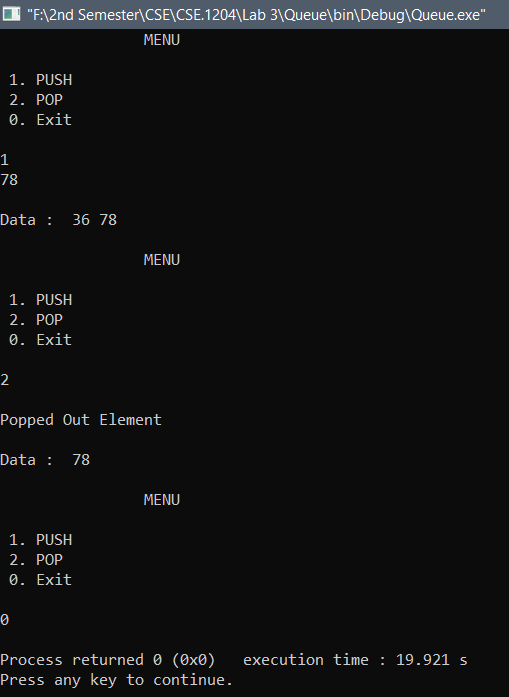
|  |
| --- |
| #include <iostream>  #include "queue.h"  using namespace std;  void menu()  {  cout<<"\n\t\t"<<"MENU"<<"\n"<<endl;  cout<<" 1. PUSH"<<"\n"<<" 2. POP";  cout<<"\n"<<" 0. Exit"<<"\n"<<endl;  }  int main()  {  queue o;  int a,i,j,k;  menu();  cin>>i;  while(i!=0)  {  if(i==1)  {  cin>>a;  o.push(a);  o.show();  menu();  cin>>i;  }  else if(i==2)  {  o.pop();  o.show();  menu();  cin>>i;  }  else if(i==0)  break;  else  {  cout<<"Wrong Input"<<"\n"<<endl;  menu();  cin>>i;  }  }  return 0;  } |

1. queue.h

|  |
| --- |
| #ifndef QUEUE\_H  #define QUEUE\_H  class queue  {  int i=0,ax[100];  public:  void push(int);  void pop();  void show();  };  #endif // QUEUE\_H |

1. queue.cpp

|  |
| --- |
| #include<iostream>  #include "queue.h"  using namespace std;  void queue::push(int a)  {  ax[i]=a;  i=i+1;  cout<<"\n"<<"Data : ";  }  void queue::pop()  {  int j;  if(i>1)  {  for(j=0;j<i-1;j++)  {  ax[j]=ax[j+1];  }  ax[i]=-1;  i=i-1;  cout<<"\n"<<"Popped Out Element"<<endl;  if(i!=0)  cout<<"\n"<<"Data : ";  }  else if(i==1)  {  ax[0]=ax[i];  ax[i]=-1;  i=i-1;  cout<<"\n"<<"Popped Out Element"<<endl;  cout<<"No Elements..."<<"\n"<<endl;  }  else  cout<<"\n"<<"No Elements..."<<"\n"<<endl;  }  void queue::show()  {  int j;  for(j=0;j<i;j++)  cout<<ax[j]<<" ";  cout<<endl;  } |

**Output:**

**Conclusion :** In the lab I just completed the basic programs. I modified the programs in home.

**The End**