Shipping Company

YMY TEAM
Version 0.1
Wed May 25 2022

Table of Contents

README

A project by Cairo engineering students for programming for first graders, which is a project we are going to build a simulator for Shipping **Company** .

Project: Shipping-Company

Course Name: Data Structures and Algorithms

Course Code: CMP1040

This is an educational project for Data Structures and Algorithms, written in C++ using Visual Studio IDE. It is a simulation of a space shipping company to deliver orders to users. So the company releases its trucks and assigns cargo to them every hour.

Features

- There are 3 Modes:
  - Iteractive Mode: Writing the simulation's stages hour by hour on the console and create the output file at the end fo simulation.
  - Step By Step Mode: With every arbitary key press, the simulation move to the next stage (hour) and create the output file when the simulation's stages are finished.
  - Silent Mode: Create Output file of the simulation without showing the simulation stages on the console.
- There is an input and output files:
  - The simulator read all the information and proccess them as it needs to start simulation. [input.txt]
  - The output file contains the final statistics generated after the simulation. [output.txt]
- The simulation consists of:
  - Status of the shipping company at every hour.
  - Showing the numbers and IDs of the trucks in different states: (In Execution, In Checkup, In Avaliable) state
  - Showing the numbers of the cargos in different states: (In Execution, In Completed, In Avaliable) state

Team Members:

- Yousef Mohamed El-Said Rabia
- Yousef Mohamed Hajjaj
- Mahmoud Sobhy Rashid

Professor Supervisor: Dr. Ahmed Hamdy

Teacher Assistants Supervisor:

- Eng. Eman Hossam
- Eng. Marwa

Data Type Index

Design Unit Hierarchy

Here is a hierarchical list of all entities:

Data Type Index

Data Types List

Here are the data types with brief descriptions:

File Index

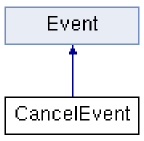
File List

Here is a list of all documented files with brief descriptions:

Data Type Documentation

CancelEvent Class Reference

Inheritance diagram for CancelEvent:



Public Member Functions

Additional Inherited Members

Member Function Documentation

void Execute () `[virtual]`

cancels a Normal cargo given its ID

Implements **Event** (*p.14*).

The documentation for this design unit was generated from the following files:

- **CancelEvent.h**
- CancelEvent.cpp

Cargo Class Reference

Public Member Functions

Private Attributes

Constructor & Destructor Documentation

Cargo (CARGO_TYPE *T*, const Time& *PT*, int *id*, float *DD*, float *LT*, double *C*)

Construct a new **Cargo** object.

Parameters

T	
PT	
id	
DD	
LT	
C	

Cargo (int *id*)

Construct a new **Cargo** object Fake cargo just for comparison with id.

Parameters

id	

Member Function Documentation

Time& Get_DT ()

Get The Delivery time.

Returns

Time&

Time& Get_Preparation_Time ()

Get Preparation **Time**.

Returns

Time&

int Get_Truck_ID ()

Get The truck carrying the cargo ID.

Returns

int

Time& Get_WT ()

Get The Wait time.

Returns

Time&

double GetCost () const

Get the Cost of cargo.

Returns

double

float GetDistance () const

Get the Distance of cargo.

Returns

float

int GetID () const

Get cargo id.

Returns

int

float GetLU_Time () const

Get the loud time of cargo.

Returns

float

Time& GetPrepTime ()

Get the Prep **Time** of cargo.

Returns

Time&

CARGO_TYPE GetType () const

Get the Type of cargo.

Returns

CARGO_TYPE

bool operator== (Cargo* *ptr*)

overloading == operator

Parameters

ptr	

Returns

true || false

void PromoteToVip (double *ExtraMoney*)

Add Extra Money to Normal cargo when Promote To Vip.

Parameters

ExtraMoney	

void Set_DT (Time *t*)

Set The Delivery time.

Parameters

Time	

void Set_Truck_ID (int *id*)

Set The truck carrying the cargo ID.

Parameters

id	

void Set_WT (int *t*)

Set The Wait time.

Parameters

int	t

Field Documentation

float Delivery_Distance `[private]`

Delivery_Distance in Km.

Time Preparation_Time `[private]`

Time(day:hour) at which the cargo is ready to be loaded.

The documentation for this design unit was generated from the following files:

- **Cargo.h**
- Cargo.cpp

Company Class Reference

Public Member Functions

Private Member Functions

Private Attributes

Constructor & Destructor Documentation

Company ()

Construct a new **Company** object.

~Company ()

Destroy the **Company** object.

Member Function Documentation

void Auto_Promotion ()

Promote cargos that exceeds a certain waiting time.

void check_checkup_list () `[private]`

Utility functions.

void Output_Console ()

function print data on console

int rest_in_waiting (Cargo* *car*)

time of rest in waiting

Parameters

*Cargo**	car

Returns

int

void Sim_Manager (SIM_MODE *Mode*)

Simulation Manager take simulation mode and to the stable operation.

Parameters

SIM_MODE	Mode

void Statistics_File (int *Delivered*, string & *text*)

Statistics are collected at the end and sent to me to put in the output file.

Parameters

int	Delivered
string	text

bool write_output_file ()

prepairs the output file at the end of the simulation

Returns

true || false

Field Documentation

int VIP_Cargos_count`[private]`

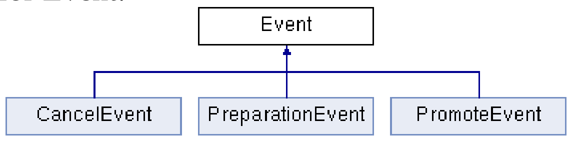
Number of cargos in each list.

The documentation for this design unit was generated from the following files:

- **Company.h**
- Company.cpp

Event Class Reference

Inheritance diagram for Event:



Public Member Functions

Protected Attributes

Private Attributes

Constructor & Destructor Documentation

virtual ~Event () `[virtual]`

Destroy the **Event** object.

Member Function Documentation

virtual void Execute () `[pure virtual]`

pure virtual function, definition is different for each event class

Implemented in **CancelEvent** (*p.7*), and **PreparationEvent** (*p.17*).

Time& getTime ()

Get the **Time** object.

Returns

Time&

The documentation for this design unit was generated from the following file:

- **Event.h**

LinkedList<Type> Class Template Reference

Public Member Functions

Private Attributes

The documentation for this design unit was generated from the following file:

- **LinkedList.h**

Node<Type> Class Template Reference

Public Member Functions

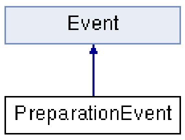
Private Attributes

The documentation for this design unit was generated from the following file:

- **Node.h**

PreparationEvent Class Reference

Inheritance diagram for PreparationEvent:



Public Member Functions

Private Attributes

Additional Inherited Members

Member Function Documentation

void Execute () `[virtual]`

prepares a cargo and adds it to the right waiting list based on its type

Implements **Event** (*p.14*).

The documentation for this design unit was generated from the following files:

- **PreparationEvent.h**
- PreparationEvent.cpp

PriNode<Type> Class Template Reference

Public Member Functions

Private Attributes

The documentation for this design unit was generated from the following file:

- PriNode.h

PriQueue<Type> Class Template Reference

Public Member Functions

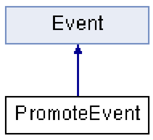
Private Attributes

The documentation for this design unit was generated from the following file:

- **PriQueue.h**

PromoteEvent Class Reference

Inheritance diagram for PromoteEvent:



Public Member Functions

Private Attributes

Additional Inherited Members

Member Function Documentation

void Execute () `[virtual]`

Implements **Event** (*p.14*).

The documentation for this design unit was generated from the following files:

- **PromoteEvent.h**
- PromoteEvent.cpp

Queue<Type> Class Template Reference

Public Member Functions

Private Attributes

The documentation for this design unit was generated from the following file:

- Queue.h

Stack<T> Class Template Reference

Public Member Functions

Private Member Functions

Private Attributes

The documentation for this design unit was generated from the following file:

- Stack.h

Time Class Reference

Public Member Functions

Private Attributes

The documentation for this design unit was generated from the following files:

- **Time.h**
- Time.cpp

Truck Class Reference

Public Member Functions

Private Attributes

Constructor & Destructor Documentation

Truck (int *id*, TRUCK_TYPE *T*, int *TC*, float *MT*, int *j*, float *S*)

Construct a new **Truck** object.

Parameters

id	
T	
TC	
MT	
j	
S	

Member Function Documentation

void DecrementJTC ()

decrement the counter after each journey

Time get_finish_point ()

Get the **Truck** finish point.

Returns

Time

float Get_nearest_dis ()

gets the distance of the nearest cargo in container (top)

Returns

float

Time Get_nearest_stop ()

Get the nearest destination time.

Returns

Time

int GetCapacity () const

Get the **Truck** Capacity.

Returns

int

int GetContainer_count ()

gets the number of cargos in container

Returns

int

float GetDeliveryInterval ()

Get the Delivery Interval object.

Returns

float

int GetID () const

Get ID.

Returns

int

int GetJTC ()

get the 'journeys till checkup' counter value

Returns

int

float GetMaintenanceTime () const

Get the **Truck** Maintenance **Time**.

Returns

float

float GetSpeed () const

Get the **Truck** Speed.

Returns

float

TRUCK_TYPE GetType () const

Get the TRUCK TYPE.

Returns

TRUCK_TYPE

void load (Cargo* *x*, float *delivery_time*)

load cargo into conatiner

Parameters

delivery_time	

void restore_JTC ()

restore 'journeys till checkup' counter with the original "J"

void set_DInterval ()

Set the **Truck** DInterval.

void set_finish_point (const Time& *t*)

set the time at which the checkup finishes

void set_nearest_stop (Time *t*, float *x*)

set the nearest destination time (delivery or return)

Cargo* unload ()

unload cargo from the container

Returns

Cargo*

float utilization (Time& *Sim_Time*)

Calc Utilization percentage.

Parameters

Sim_Time	

Returns

float

Field Documentation

Time AT `[private]`

Active **Time**.

float Delivery_Distance `[private]`

distance of the furthest cargo in container

float Delivery_Interval `[private]`

Time to deliver all cargos & comeback, Calculated.

Time finish_point `[private]`

when the checkup finishes

int J `[private]`

journeys untill checkup

int Journeys_Till_Check `[private]`

counter for journeys untill checkup

float Maintenance_Time `[private]`

hours

int move_counter `[private]`

initialized with the highest load_time cargo in the container, once it reaches 0 the truck moves.

int N `[private]`

total delivery journeys of this truck

float Nearest_dis `[private]`

distance for the nearest cargo

Time Nearest_stop `[private]`

the delivery time for the nearsest cargo

int TDC `[private]`

total cargos delivered by this truck

int Truck_Capacity `[private]`

# of cargos

The documentation for this design unit was generated from the following files:

- **Truck.h**
- Truck.cpp

UI Class Reference

Public Member Functions

Member Function Documentation

int getIntger ()

get Intger from user

string getString ()

get String from user

void print (string *s*)

print string

The documentation for this design unit was generated from the following files:

- **UI.h**
- UI.cpp

File Documentation

CancelEvent.h File Reference

cancels a normal cargo

Data Structures

- class **CancelEvent**

Detailed Description

cancels a normal cargo

Version

0.1

Copyright

Copyright secured by YMY Team(c) 2022

CancelEvent.h

Go to the documentation of this file.

```
#pragma once
#include "Event.h"
class CancelEvent :
    public Event
{
public:
    CancelEvent(Company* p, const Time&, int);
    void Execute();
};

```

Cargo.h File Reference

Cargo Class.

Data Structures

- class **Cargo**

Detailed Description

Cargo Class.

Version

0.1

Copyright

Copyright secured by YMY Team(c) 2022

Function Documentation

ostream& operator<< (ostream& *out*, const Cargo* *c*)

overloading << operator

Parameters

ostream&	out
*Cargo**	c

Returns

ostream&

Cargo.h

Go to the documentation of this file.1

```
#pragma once
#include"Def.h"
#include "Time.h"
#include <string>
#include <iostream>

class Cargo
{
private:
    Time Preparation_Time;
    Time WT;
    Time DT;
    float Load_Unload_Time;
    CARGO_TYPE Type;
    float Delivery_Distance;
    double Cost;
    int ID;
    int Truck_ID;

public:
    Cargo(CARGO_TYPE T, const Time& PT, int id, float DD, float LT, double C);
    Cargo(int id);
    float GetDistance() const;
    double GetCost() const;
    float GetLU_Time() const;
    CARGO_TYPE GetType() const;
    Time& GetPrepTime();
    void PromoteToVip(double ExtraMoney);
    bool operator==(Cargo* ptr);
    int GetID() const;
    void Set_Truck_ID(int id);
    int Get_Truck_ID();
    void Set_DT(Time t);
    void Set_WT(int t);
    Time& Get_DT();
    Time& Get_WT();
    Time& Get_Preparation_Time();


};
ostream& operator<<(ostream& out, const Cargo* c);



```

Company.h File Reference

Manager of all operations that occur in the program.

Data Structures

- class **Company**

Detailed Description

Manager of all operations that occur in the program.

Version

0.1

Copyright

Copyright secured by YMY Team(c) 2022

Company.h

Go to the documentation of this file.1

```
#pragma once
#include "Def.h"
#include "Cargo.h"
#include "Truck.h"
#include "UI.h"
#include "Queue.h"
#include "PriQueue.h"
#include "LinkedList.h"
#include <fstream>

class Event;

class Company
{
    UI* ui_p;
    Time Sim_Time;
    ifstream Loaded;
    //--------------------------------------------------------

    //Cargos
    PriQueue<Cargo*> W_V_C;     //waiting vip cargos
    Queue<Cargo*> W_S_C;        //waiting special cargos
    LinkedList<Cargo*> W_N_C;   //waiting normal cargos
    Queue<Cargo*> Delivered_cargo;

    //--------------------------------------------------------

    //Trucks
    Queue<Truck*> empty_VIP; //avail. VIP trucks
    Queue<Truck*> empty_Special; //avail. Special trucks
    Queue<Truck*> empty_Normal;  //avail. Normal trucks
    Queue<Truck*> Check_up_Normal;  //Normla trucks in check up
    Queue<Truck*> Check_up_Special;  //Special trucks in Check up
    Queue<Truck*> Check_up_VIP;  //VIP trucks in check up
    PriQueue<Truck*> Moving_truck;

    Queue<Event*> Event_List;

    //---------------------------------------------------------

    int MaxWait;
    int AutoPro;
    int Num_of_events;

    int nCap, sCap, vCap;
    //Numbers of Trucks in each list

    int VIP_Trucks_count;
    int Normal_Trucks_count;
    int Special_Trucks_count;
    int Assigned_Trucks_count;
    int InCheck_Trucks_count;
    int Total_Trucks_count;

    //---------------------------------------------------------



    int VIP_Cargos_count;
    int Normal_Cargos_count;
    int Special_Cargos_count;
    int Moving_Cargos_count;
    int Delivered_Cargos_count;
    int Total_Cargos_count;
    int Num_Promoted_cargos;

    //----------------------------------------------------------
    float auto_promoted_count;
    int cancelled;

    //----------------------------------------------------------
    Truck* Loading_Normal;
    Truck* Loading_Special;
```

```cpp
    Truck* Loading_VIP;

    //--------------------------------------------------------
    void check_checkup_list();
    void check_to_available(Truck*&); //moves a truck from checkup to available
    void move_to_available(Truck*); //moves a truck from moving to available
    void move_to_checkup(Truck*); //moves a truck from moving to checkup

    Truck* Pick_VIP_Truck(); //picks the appropriate truck from loading VIP cargos
    Truck* Pick_Normal_Truck(); //picks the appropriate truck from loading Normal cargos
    Truck* Pick_Special_Truck();//picks the appropriate truck from loading Special cargos
    bool load_VIP();
    bool load_Normal();
    bool load_Special();
    bool load_MaxW();
    bool Need_Checkup(Truck*); //checks on a returning truck if it needs maintainence
    bool in_working(Time T);
    void Loading_count(int&, int&);

public:
    Company();
    void Start_Simuulation();
    void Working_Hours();
    void Truck_Controller(); //controls the transition of trucks between different lists
    void Off_Hours();

    // Reading data function
    void execute_mode(SIM_MODE);
    bool readFile(string);
    bool write_output_file();
    void Statistics_File(int Delivered,string & text);
    int rest_in_waiting(Cargo* car);
    Time& get_Sim_Time() ;
    Time& get_Nearest_Event_Time();
    Event* get_Nearest_Event();
    void Advance_Sim_Time(int = 1);
    void Auto_Promotion();
    // Simulation Functions
    void assign_cargo();
    void check_completed_cargo();
    void increment_cancelled();
    void Deliver_cargos(); //deliver cargos when reached its destination

    //Printing Functions
    void Print_Sim_Time();
    void InteractivePrinting();
    void StepByStepPrinting();
    void SilentPrinting();
    void Sim_Manager(SIM_MODE Mode);

    //----------------------------------------
    void print_W_V_C();
    void print_W_S_C();
    void print_W_N_C();

    //----------------------------------------
    void print_check_up_v_trucks();
    void print_check_up_s_trucks();
    void print_check_up_n_trucks();
    void print_empty_VIP();
    void print_empty_Normal();
    void print_empty_Special();
    //---------------------------------------
    bool Events_empty();

    void AddCargo(Cargo*);
    void Waiting_To_Delivered();

    bool Upgrade_Normal_Cargo(int id,int extra_money=0);
    bool Cancel_Normal_Cargo(int id);
    void Move_Trucks();         //checks for truck movement and calls Move_Truck accordingly
    void Move_Truck(Truck*& t); //actually moves the truck, adding it to the Moving_Truck queue

    bool All_Delivered();       //checks that all waiting and moving lists are empty

    //------------------------------------------------------------
    SIM_MODE get_sim_mode();
    void Output_Console();


    //Destructor
    ~Company(){}


};


```

Def.h File Reference

Some Definitions and enums.

Detailed Description

Some Definitions and enums.

Version

0.1

Copyright

Def.h

```
Go to the documentation of this file.
#pragma once

enum class SIM_MODE
{
    INTERACTIVE,
    STEP_BY_STEP,
    SILENT
};


enum class TRUCK_TYPE {
    VIP,
    SPECIAL,
    NORMAL
};

enum class CARGO_TYPE {
    VIP,
    SPECIAL,
    NORMAL
};



```

Event.h File Reference

Abstract class and parent of another children classes.

Data Structures

- class **Event**

Detailed Description

Abstract class and parent of another children classes.

Version

0.1

Copyright

Copyright secured by YMY Team(c) 2022

Event.h

Go to the documentation of this file.

```
#pragma once
#include "Time.h"
#include "Company.h"

class Event
{
    UI* UI_P;

protected:
    Time ET;
    int ID;
    Company* cPtr;
public:
    Event(Company* p, const Time& T, int id)
    {
        cPtr = p;
        ID = id;
        ET = T;
    }

    virtual void Execute() = 0;
    Time& getTime()
    {
        return ET;
    }
    virtual ~Event()
    {
        delete UI_P;
    }
};
```

LinkedList.h File Reference

LinkedList data structure.

Data Structures

- class **LinkedList<Type>**

Detailed Description

LinkedList data structure.

Version

0.1

Copyright

LinkedList.h

Go to the documentation of this file.

```
#pragma once
#include"Node.h"
#include <iostream>
using namespace std;
#include "Cargo.h"
#include "Truck.h"
template<class Type>
class LinkedList
{
private:
    int count;
    Node<Type>* First;
    Node<Type>* End;
public:
    LinkedList()
    {
        First = NULL;
        End = NULL;
        count = 0;
    }
    void InsertBegin(Type item)
    {
        Node<Type>* temp;
        temp = new Node<Type>;
        temp->set_item(item);
        if (IsEmpty())
            End = temp;
        else
        temp->set_next(First);
        First = temp;
        count++;
    }
    bool InsertIndex(Type item, int index)
    {
        if (index <= count && index >= 0)
        {
            if (index == 0)
                InsertBegin(item);
            else if (index == count)
                InsertEnd(item);
            else
            {
                Node<Type>* temp;
                temp = new Node<Type>;
                Node<Type>* cur = First;
                for (int i = 0; i < index - 1; i++)
                    cur = cur->get_next;
                temp->set_next(cur->get_next());
                cur->set_next(temp);
                count++;
            }
            return true ;
        }
        else
        return false;
    }
    void InsertEnd(Type item)
    {
        Node<Type>* temp;
        temp = new Node<Type>;
        temp->set_item(item);
        if (IsEmpty())
            First = temp;
        else
            End->set_next(temp);
        End = temp;
        count++;
    }
    Type getFirst()
    {
        if (!IsEmpty())
            return First->get_item();
```

```cpp
        else
            return NULL;
    }

    Type getEnd()
    {
        if (!IsEmpty())
            return End->get_item();
        else
            return NULL;
    }

    bool IsEmpty()
    {
        return !count;
    }

    int GetCount()
    {
        return count;
    }
     bool Find_Remove(Type val,Type& x)
     {
         Node<Type>* prev = NULL;
         Node<Type>* ptr = First;
         while (ptr)
         {
             if (*(ptr->get_item()) == val)
             {
                 if (prev == NULL)
                 {
                     x = ptr->get_item();
                     First = First->get_next();
                 }
                 else
                 {
                     x = ptr->get_item();
                     prev->set_next(ptr->get_next());
                 }
                 count--;
                 return true;
             }
             prev = ptr;
             ptr = ptr->get_next();
         }
         return false;
     }

     bool removeFirst(Type& x)
     {
         if (count == 0)
             return false;

         Node<Type>* ptr;
         ptr = First;
         if (count == 1)
         {
             First = NULL;
             End = NULL;
         }
         else
         {
             First= First->get_next();
         }
         x = ptr->get_item();
         count--;
         return true;
     }
     void print()
     {
         Node<Type>* temp = First;
         while (temp)
         {
             cout << (temp->get_item());
             temp = temp->get_next();
             if (temp)
                 cout << ',';
        }
    }

};
```

Main.cpp File Reference

Main of Project.

Detailed Description

Main of Project.

Version

0.1

Copyright

Node.h File Reference

Data Structures

- class **Node<Type>**

Detailed Description

Version

0.1

Copyright

Copyright secured by YMY Team(c) 2022

Node.h

Go to the documentation of this file.1

```
#pragma once
template<class Type>
class Node
{
private:
    Type item;
    int Priority;
    Node<Type>* next;
public:
    Node() :next(nullptr)
    {
    }
    Node(const Type anitem)
    {
        item = anitem;
        next = nullptr;
    }
    Node(const Type anitem, Node<Type>* nextptr)
    {
        item = anitem;
        next = nextptr;
    }
    void set_item(const Type anitem) { item = anitem; }
    void set_next(Node<Type>* nextptr) { next = nextptr; }

    Type get_item()const
    {
        return item;
    }
    Node<Type>* get_next()const { return next; }


};
```

PreparationEvent.h File Reference

Children Class of **Event**.

Data Structures

- class **PreparationEvent**

Detailed Description

Children Class of **Event**.

Version

0.1

Copyright

Copyright secured by YMY Team(c) 2022

PreparationEvent.h

Go to the documentation of this file.

```
#pragma once
#include "Event.h"
#include "Time.h"
#include "Def.h"

class PreparationEvent :
    public Event
{
    CARGO_TYPE type;
    float dist, loadTime, cost;
public:
    PreparationEvent(Company* p, CARGO_TYPE, const Time&, int, float, float, float);
    void Execute();
};

```

PriNode.h

```
#pragma once
template<class Type>
class PriNode
{

    private:
        Type item;
        int Priority;
        PriNode<Type>* next;
     public:
         PriNode() :next(nullptr)
         {
             Priority = 0;
         }
         PriNode(const Type anitem)
         {
             item = anitem;
             next = nullptr;
             Priority = 0;
         }
         PriNode(const Type anitem, PriNode<Type>* nextptr)
         {
             item = anitem;
             next = nextptr;
             Priority = 0;
         }
         void set_item(const Type anitem) { item = anitem; }
         void set_next(PriNode<Type>* nextptr) { next = nextptr; }
         void set_priority(int p)
         {
             Priority = p;
         }

         Type get_item()const { return item; }
         PriNode<Type>* get_next()const { return next; }
         int get_priority()
         {
             return Priority;
         }

     };



```

PriQueue.h File Reference

PriQueue Data structure.

Data Structures

- class **PriQueue<Type>**

Detailed Description

PriQueue Data structure.

Version

0.1

Copyright

Copyright secured by YMY Team(c) 2022

PriQueue.h

Go to the documentation of this file.

```
#pragma once
#include "PriNode.h"
#include<iostream>

template<class Type>
class PriQueue
{

private:
    PriNode<Type>* front;
    int count;
    PriNode<Type>* rear;
public:
    PriQueue()
    {
        front = nullptr;
        rear = nullptr;
        count = 0;
    }
    bool EnQueue(Type item, float priority = 0)
    {
        PriNode<Type>* temp;
        PriNode<Type>* cur = front;
        temp = new PriNode<Type>;
        temp->set_item(item);
        temp->set_priority(priority);
        if (QueueEmpty())
        {
            front = temp;
            rear = temp;

        }
        else if (front->get_priority() < priority)
        {
            temp->set_next(front);
            front = temp;
        }
        else
        {
            while (cur->get_next() != NULL && cur->get_next()->get_priority() >=
priority)
                cur = cur->get_next();
            if (!cur->get_next())
                rear = temp;
            temp->set_next(cur->get_next());
            cur->set_next(temp);

        }

        count++;
        return true;

    }
    bool DeQueue(Type& x)
    {
        if (count == 0)
            return false;
        PriNode<Type>* delptr;
        delptr = front;
        if (count == 1)
        {
            rear = NULL;
            front = NULL;
        }
        else
        {
            front = front->get_next();
        }
        delptr->set_next(nullptr);
        x = delptr->get_item();
        count--;
        return true;
    }
    bool QueueEmpty()
    {
        return !count;
    }
    void DistroyQueue()
    {
        Type x;
        int c = count;
        for (int i = 0; i < c; i++)
            DeQueue(x);
    }
    PriNode<Type>* GetFront()
    {
        return front;
    }
    PriNode<Type>* GetRear()
    {
        return rear;
    }
    int GetCount()
    {
         return count;
     }
     Type Peek()
     {
         if (!QueueEmpty())
             return front->get_item();
         else
             return NULL;
     }

     ~PriQueue()
     {
         DistroyQueue();
     }

     void print()
     {
         PriNode<Type>* temp = front;
         while (temp)
         {
             cout << (temp->get_item());
             temp = temp->get_next();
             if (temp)
                 cout << ',';
         }
     }
};

```

PromoteEvent.h File Reference

Children Class of **Event**.

Data Structures

- class **PromoteEvent**

Detailed Description

Children Class of **Event**.

Version

0.1

Copyright

PromoteEvent.h

Go to the documentation of this file.1

```
#pragma once
#include "Event.h"
class PromoteEvent :
    public Event
{
    //money used to promote the cargo, added to its original cost
    float ExtraMoney;
public:
    PromoteEvent(Company* p, const Time&, int, float);
    //promotes a Normal cargo to VIP given its ID
    void Execute();
};

```

Queue.h

```cpp
#pragma once
#include"Node.h"
#include <iostream>
using namespace std;

template<class Type>
class Queue
{
private:
    Node<Type>* front;
    int count;
    Node<Type>* rear;
public:
    Queue()
    {
        front = nullptr;
        rear = nullptr;
        count = 0;
    }
    bool EnQueue(Type item)
    {
        Node<Type>* temp;
        temp = new Node<Type>;
        temp->set_item(item);
        if (QueueEmpty())
            front = temp;
        else
            rear->set_next(temp);

        rear = temp;
        count++;
        return true;

    }
    bool DeQueue(Type& x)
    {
        if (count == 0)
            return false;
        Node<Type>* delptr;
        delptr = front;
        if (count == 1)
        {
            rear = NULL;
            front = NULL;
        }
        else
        {
            front = front->get_next();
        }
        delptr->set_next(nullptr);
        x = delptr->get_item();
        count--;
        return true;
    }
    bool QueueEmpty()
    {
        return !count;
    }
    void DistroyQueue()
    {
        Type x;
        int c = count;
        for (int i = 0; i < c; i++)
            DeQueue(x);
    }
    Node<Type>* GetFront()
    {
        return front;
    }
    Node<Type>* GetRear()
    {
        return rear;
    }
    int GetCount()
    {
        return count;
    }
    Type Peek()
    {
        if (!QueueEmpty())
            return front->get_item();
        else
            return NULL;
    }

    ~Queue()
    {
        DistroyQueue();
    }

    void print()
    {
        Node<Type>* temp = front;
        while (temp)
        {
            cout << temp->get_item();
            temp = temp->get_next();
            if (temp)
                cout << ',';
         }
     }
};
```

Stack.h

```cpp
#include"Node.h"
#include <cassert>

template <class T>
class Stack
{
    Node<T>* Head;

    void copyStack(const Stack<T>& R) {
         if (R.Head == NULL)
         {
             IntializeStack();
             return;
         }
         if (R.Head == Head)
             return;
         IntializeStack();
         Node<T>* ptr;
         Node<T>* Rptr;
         Head = new Node<T>;
         Head->setitem(R.Head->getitem());
         ptr = Head;
         Rptr = R.Head->getnext();
         while (Rptr) {
             ptr->getnext() = new Node<T>;
             ptr = ptr->getnext();
             ptr->setitem(Rptr->getitem());
             Rptr = Rptr->getnext();

         }
         ptr->setnext(NULL);
     }
 public:
     Stack() {
         Head = NULL;
     }
     Stack(const Stack<T>& R) {
         Head = NULL;
         copyStack(R);

     }
     void IntializeStack() {
         if (Head == NULL)
             return;
         Node<T>* ptr = Head;
         Node<T>* nextptr;


         while (ptr) {
             nextptr = ptr->getnext();
             delete ptr;
             ptr = nextptr;
         }
         Head = NULL;
     }
     bool IsEmptyStack() {
         return (Head == NULL);
     }

     const Stack<T>& operator =(const Stack<T>& R) {
         copyStack();
         return *this;

     }
     void Push(T data) {
         Node <T>* ptr = new Node<T>(data);
         if (Head == NULL)
         {
             Head = ptr;
             Head->setnext(NULL);
             return;
         }

```

```
        ptr->setnext(Head);
        Head = ptr;
    }

     bool Pop() {
        if (IsEmptyStack())
            return false;
        Node <T>* ptr = Head;
        Head = Head->getnext();
        delete ptr;
        return true;
    }
    const T& Top() {
        assert(Head != NULL);
        return Head->getitem();
    }
    ~Stack() {
        IntializeStack();
    }
};

```

Time.h File Reference

Class responsible for time.

Data Structures

- class **Time**

Detailed Description

Class responsible for time.

Version

0.1

Copyright

Copyright secured by YMY Team(c) 2022

Time.h

Go to the documentation of this file.

```cpp
#pragma once
#include <string>
#include "UI.h"
using namespace std;

class Time
{
    UI UI_P;
    int hour;
    int day;
public:
    Time();
    Time(int d , int h);
    Time(string x);
    Time(int h);
    void setTime(int h);
    void setTime(string x);
    void AdvanceTime(int value);
    void printTime();
    int getHour();
    int getDay();
    int Time_In_Hours();   //returns (24*day+hour) which is the total number of hours
    string Time_to_print();
    bool operator==(const Time&);
    bool operator>=(const Time&);
    int operator-(const Time&);
    Time operator +(float x);
    ~Time();
};

```

Truck.h File Reference

Truck Class.

Data Structures

- class **Truck**

Detailed Description

Truck Class.

Version

0.1

Copyright

Copyright secured by YMY Team(c) 2022

Function Documentation

ostream& operator<< (ostream& *out*, Truck* *t*)

overloading << operator

Parameters

ostream&	out
*Truck**	c

Returns

ostream&

Truck.h

Go to the documentation of this file.

```cpp
#pragma once
#include<string>
#include"Def.h"
#include "Cargo.h"
#include "PriQueue.h"

using namespace std;
class Truck
{
private:

    UI* ui_p;
    PriQueue<Cargo*>container;
    TRUCK_TYPE Type;
    int Truck_Capacity;
    float Maintenance_Time;
    float Speed; //Km/h
    int J;
    int Journeys_Till_Check;
    float Delivery_Interval;
    float Delivery_Distance;
    Time Nearest_stop;
    float Nearest_dis;
    int ID;
    int move_counter;
    Time finish_point;
    Time AT;
    int TDC;
    int N;
    Time moving_time;

public:
    Truck(int id, TRUCK_TYPE T, int TC, float MT, int j, float S);
    TRUCK_TYPE GetType() const;
    int GetCapacity() const;
    void set_finish_point(const Time&);
    Time get_finish_point();
    float GetMaintenanceTime() const;
    float Get_nearest_dis();
    float GetSpeed() const;
    float GetDeliveryInterval();
    void set_nearest_stop(Time, float);
    Time Get_nearest_stop();
    int GetJTC();
    int GetContainer_count();
    void restore_JTC();
    void DecrementJTC();
    int GetID() const;
    void set_DInterval();
    void load(Cargo*, float delivery_time);
    Cargo* unload();
    void count_down();
    int get_move_counter();
    void Set_AT(int h);
    void inc_TDC();
    void inc_N();
    int Get_TDC();
    int Get_N();
    Time Get_AT();
    float utilization(Time& Sim_Time);
    void print_container();
    void print();
    void Set_moving_time(Time& Sim_Time);
    Time get_moving_time();

};
ostream& operator<<(ostream& out, Truck* t);
```

UI.h File Reference

Truck Class.

Data Structures

- class **UI**

Detailed Description

Truck Class.

Version

0.1

Copyright

UI.h

Go to the documentation of this file.

```cpp
#pragma once
#include "Def.h"
#include <iostream>
#include <string>
using namespace std;

class UI
{

public:
    UI();
    void print(string);
    int getIntger();
    string getString();
};

```