

# **Shipping Company**

YMY TEAM  
Version 0.1  
Wed May 25 2022



# Table of Contents

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:
  - Interactive 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 arbitrary 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 Aavailable) state
  - Showing the numbers of the cargos in different states: (In Execution, In Completed, In Aavailable) 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:

Cargo .....	8
Company .....	12
Event .....	14
CancelEvent .....	7
PreparationEvent .....	17
PromoteEvent .....	20
LinkedList<Type> .....	15
LinkedList<Cargo*> .....	15
Node<Type> .....	16
Node<Cargo* > .....	16
Node<Event* > .....	16
Node<T> .....	16
Node<Truck* > .....	16
PriNode<Type> .....	18
PriNode<Cargo* > .....	18
PriNode<Truck* > .....	18
PriQueue<Type> .....	19
PriQueue<Cargo*> .....	19
PriQueue<Truck*> .....	19
Queue<Type> .....	21
Queue<Cargo*> .....	21
Queue<Event*> .....	21
Queue<Truck*> .....	21
Stack<T> .....	22
Time .....	23
Truck .....	24
UI .....	29

# Data Type Index

## Data Types List

Here are the data types with brief descriptions:

<b>CancelEvent</b>	7
<b>Cargo</b>	8
<b>Company</b>	12
<b>Event</b>	14
<b>LinkedList&lt;Type&gt;</b>	15
<b>Node&lt;Type&gt;</b>	16
<b>PreparationEvent</b>	17
<b>PriNode&lt;Type&gt;</b>	18
<b>PriQueue&lt;Type&gt;</b>	19
<b>PromoteEvent</b>	20
<b>Queue&lt;Type&gt;</b>	21
<b>Stack&lt;T&gt;</b>	22
<b>Time</b>	23
<b>Truck</b>	24
<b>UI</b>	29



# File Index

## File List

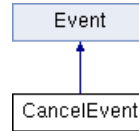
Here is a list of all documented files with brief descriptions:

<b>CancelEvent.h (Cancels a normal cargo )</b>	30
<b>Cargo.h (Cargo Class )</b>	32
<b>Company.h (Manager of all operations that occur in the program )</b>	34
<b>Def.h (Some Definitions and enums )</b>	38
<b>Event.h (Abstract class and parent of another children classes )</b>	40
<b>LinkedList.h (LinkedList data structure )</b>	42
<b>Main.cpp (Main of Project )</b>	46
<b>Node.h</b>	47
<b>PreparationEvent.h (Children Class of Event )</b>	49
<b>PriNode.h</b>	51
<b>PriQueue.h (PriQueue Data structure )</b>	52
<b>PromoteEvent.h (Children Class of Event )</b>	55
<b>Queue.h</b>	57
<b>Stack.h</b>	59
<b>Time.h (Class responsible for time )</b>	61
<b>Truck.h (Truck Class )</b>	63
<b>UI.h (Truck Class )</b>	65

# 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

<i>T</i>	
<i>PT</i>	
<i>id</i>	
<i>DD</i>	
<i>LT</i>	
<i>C</i>	

**Cargo** (int *id*)

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

### Parameters

<i>id</i>	
-----------	--

---

## 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**

<i>ptr</i>	
------------	--

**Returns**

true || false

**void PromoteToVip (double *ExtraMoney*)**

Add Extra Money to Normal cargo when Promote To Vip.

**Parameters**

<i>ExtraMoney</i>	
-------------------	--

**void Set\_DT (Time *t*)**

Set The Delivery time.

**Parameters**

<i>Time</i>	
-------------	--

**void Set\_Truck\_ID (int *id*)**

Set The truck carrying the cargo ID.

**Parameters**

<i>id</i>	
-----------	--

**void Set\_WT (int *t*)**

Set The Wait time.

### Parameters

<i>int</i>	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

<i>Cargo*</i>	car
---------------	-----

#### Returns

int

### void Sim\_Manager (SIM\_MODE Mode)

Simulation Manager take simulation mode and to the stable operation.

#### Parameters

<i>SIM_MODE</i>	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

<i>int</i>	Delivered
<i>string</i>	text

**bool write\_output\_file ()**

prepares 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.

---

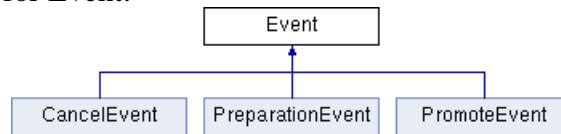
**The documentation for this design unit was generated from the following files:**

- **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**

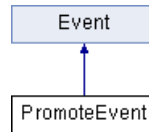
---

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

<i>id</i>	
<i>T</i>	
<i>TC</i>	
<i>MT</i>	
<i>j</i>	
<i>S</i>	

---

### 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**

<i>delivery_time</i>	
----------------------	--

**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

<i>Sim_Time</i>	
-----------------	--

### 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 nearest 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.

```
7 #pragma once
8 #include "Event.h"
9 class CancelEvent :
10     public Event
11 {
12 public:
13     CancelEvent(Company* p, const Time&, int);
17     void Execute();
18 };
19
```

# 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

<i>ostream&amp;</i>	out
<i>Cargo*</i>	c

### Returns

ostream&

## Cargo.h

Go to the documentation of this file.

```
7 #pragma once
8 #include "Def.h"
9 #include "Time.h"
10 #include <string>
11 #include <iostream>
12
13 class Cargo
14 {
15 private:
16     Time Preparation_Time;
17     Time WT;
18     Time DT;
19     float Load_Unload_Time;
20     CARGO_TYPE Type;
21     float Delivery_Distance;
22     double Cost;
23     int ID;
24     int Truck_ID;
25
26 public:
27     Cargo(CARGO_TYPE T, const Time& PT, int id, float DD, float LT, double C);
28     Cargo(int id);
29     float GetDistance() const;
30     double GetCost() const;
31     float GetLU_Time() const;
32     CARGO_TYPE GetType() const;
33     Time& GetPrepTime();
34     void PromoteToVip(double ExtraMoney);
35     bool operator==(Cargo* ptr);
36     int GetID() const;
37     void Set_Truck_ID(int id);
38     int Get_Truck_ID();
39     void Set_DT(Time t);
40     void Set_WT(int t);
41     Time& Get_DT();
42     Time& Get_WT();
43     Time& Get_Preparation_Time();
44
45 };
46
47 ostream& operator<<(ostream& out, const Cargo* c);
48
49
50
51
52
53
```

# 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

```
7 #pragma once
8 #include "Def.h"
9 #include "Cargo.h"
10 #include "Truck.h"
11 #include "UI.h"
12 #include "Queue.h"
13 #include "PriQueue.h"
14 #include "LinkedList.h"
15 #include <fstream>
16
17 class Event;
18
19 class Company
20 {
21     UI* ui_p;
22     Time Sim_Time;
23     ifstream Loaded;
24     //-----
25
26     //Cargos
27     PriQueue<Cargo*> W_V_C;    //waiting vip cargos
28     Queue<Cargo*> W_S_C;    //waiting special cargos
29     LinkedList<Cargo*> W_N_C; //waiting normal cargos
30     Queue<Cargo*> Delivered_cargo;
31
32     //-----
33
34     //Trucks
35     Queue<Truck*> empty_VIP; //avail. VIP trucks
36     Queue<Truck*> empty_Special; //avail. Special trucks
37     Queue<Truck*> empty_Normal; //avail. Normal trucks
38     Queue<Truck*> Check_up_Normal; //Normal trucks in check up
39     Queue<Truck*> Check_up_Special; //Special trucks in Check up
40     Queue<Truck*> Check_up_VIP; //VIP trucks in check up
41     PriQueue<Truck*> Moving_truck;
42
43     Queue<Event*> Event List;
44
45     //-----
46
47     int MaxWait;
48     int AutoPro;
49     int Num of events;
50
51     int nCap, sCap, vCap;
52     //Numbers of Trucks in each list
53
54     int VIP_Trucks_count;
55     int Normal_Trucks_count;
56     int Special_Trucks_count;
57     int Assigned_Trucks_count;
58     int InCheck_Trucks_count;
59     int Total_Trucks_count;
60
61     //-----
62
63
64     int VIP_Cargos_count;
65     int Normal_Cargos_count;
66     int Special_Cargos_count;
67     int Moving_Cargos_count;
68     int Delivered_Cargos_count;
69     int Total_Cargos_count;
70     int Num_Promoted_cargos;
71
72     //-----
73
74     float auto_promoted_count;
75     int cancelled;
76
77     //-----
78     Truck* Loading_Normal;
79     Truck* Loading_Special;
```

```

83     Truck* Loading_VIP;
84
85     //-----
90     void check_checkup_list();
91     void check_to_available(Truck*&); //moves a truck from checkup to available
92     void move_to_available(Truck*); //moves a truck from moving to available
93     void move_to_checkup(Truck*); //moves a truck from moving to checkup
94
95     Truck* Pick_VIP_Truck(); //picks the appropriate truck from loading VIP cargos
96     Truck* Pick_Normal_Truck(); //picks the appropriate truck from loading Normal
cargos
97     Truck* Pick_Special_Truck(); //picks the appropriate truck from loading Special
cargos
98     bool load_VIP();
99     bool load_Normal();
100     bool load_Special();
101     bool load_MaxW();
102     bool Need_Checkup(Truck*); //checks on a returning truck if it needs maintenance
103     bool in_working(Time T);
104     void Loading_count(int&, int&);
105
106 public:
111     Company();
112     void Start_Simulation();
113     void Working_Hours();
114     void Truck_Controller(); //controls the transition of trucks between different
lists
115     void Off_Hours();
116
117     // Reading data function
118     void execute_mode(SIM_MODE);
119     bool readFile(string);
125     bool write_output_file();
131     void Statistics_File(int Delivered, string & text);
138     int rest_in_waiting(Cargo* car);
139     Time& get_Sim_Time();
140     Time& get_Nearest_Event_Time();
141     Event* get_Nearest_Event();
142     void Advance_Sim_Time(int = 1);
147     void Auto_Promotion();
148     // Simulation Functions
149     void assign_cargo();
150     void check_completed_cargo();
151     void increment_cancelled();
152     void Deliver_cargos(); //deliver cargos when reached its destination
153
154     //Printing Functions
155     void Print_Sim_Time();
156     void InteractivePrinting();
157     void StepByStepPrinting();
158     void SilentPrinting();
164     void Sim_Manager(SIM_MODE Mode);
165
166     //-----
167     void print_W_V_C();
168     void print_W_S_C();
169     void print_W_N_C();
170
171     //-----
172     void print_check_up_v_trucks();
173     void print_check_up_s_trucks();
174     void print_check_up_n_trucks();
175     void print_empty_VIP();
176     void print_empty_Normal();
177     void print_empty_Special();
178     //-----
179     bool Events_empty();
180
181     void AddCargo(Cargo*);
182     void Waiting_To_Delivered();
183
184     bool Upgrade_Normal_Cargo(int id, int extra_money=0);
185     bool Cancel_Normal_Cargo(int id);
186     void Move_Trucks(); //checks for truck movement and calls Move_Truck
accordingly
187     void Move_Truck(Truck*& t); //actually moves the truck, adding it to the
Moving_Truck queue

```

```
188
189     bool All_Delivered();          //checks that all waiting and moving lists are empty
190
191     //-----
192     SIM_MODE get_sim_mode();
193     void Output_Console();
194
195
196
197
198
199
200     //Destructor
201     ~Company() {}
202
203
204
205
206
207
208 };
209
210
```



## Def.h File Reference

Some Definitions and enums.

---

### Detailed Description

Some Definitions and enums.

### Version

0.1

### Copyright

Copyright secured by YMY Team(c) 2022 \*

## Def.h

[Go to the documentation of this file.](#)<sup>1</sup>

```
7 #pragma once
8
9 enum class SIM_MODE
10 {
11     INTERACTIVE,
12     STEP_BY_STEP,
13     SILENT
14 };
15
16
17 enum class TRUCK_TYPE {
18     VIP,
19     SPECIAL,
20     NORMAL
21 };
22
23 enum class CARGO_TYPE {
24     VIP,
25     SPECIAL,
26     NORMAL
27 };
28
29
30
```

## 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.[1](#)

```
7 #pragma once
8 #include "Time.h"
9 #include "Company.h"
10
11 class Event
12 {
13     UI* UI_P;
14
15 protected:
16     Time ET;
17     int ID;
18     Company* cPtr;
19 public:
20     Event(Company* p, const Time& T, int id)
21     {
22         cPtr = p;
23         ID = id;
24         ET = T;
25     }
26
27     virtual void Execute() = 0;
28     Time& getTime()
29     {
30         return ET;
31     }
32     virtual ~Event()
33     {
34         delete UI_P;
35     }
36 };
```

# LinkedList.h File Reference

**LinkedList** data structure.

## Data Structures

- class **LinkedList**<Type>

---

## Detailed Description

**LinkedList** data structure.

## Version

0.1

## Copyright

Copyright secured by YMY Team(c) 2022

## LinkedList.h

Go to the documentation of this file.[1](#)

```
7 #pragma once
8 #include "Node.h"
9 #include <iostream>
10 using namespace std;
11 #include "Cargo.h"
12 #include "Truck.h"
13 template<class Type>
14 class LinkedList
15 {
16 private:
17     int count;
18     Node<Type>* First;
19     Node<Type>* End;
20 public:
21     LinkedList()
22     {
23         First = NULL;
24         End = NULL;
25         count = 0;
26     }
27     void InsertBegin(Type item)
28     {
29         Node<Type>* temp;
30         temp = new Node<Type>;
31         temp->set_item(item);
32         if (IsEmpty())
33             End = temp;
34         else
35             temp->set_next(First);
36         First = temp;
37         count++;
38     }
39     bool InsertIndex(Type item, int index)
40     {
41         if (index <= count && index >= 0)
42         {
43             if (index == 0)
44                 InsertBegin(item);
45             else if (index == count)
46                 InsertEnd(item);
47             else
48             {
49                 Node<Type>* temp;
50                 temp = new Node<Type>;
51                 Node<Type>* cur = First;
52                 for (int i = 0; i < index - 1; i++)
53                     cur = cur->get_next();
54                 temp->set_next(cur->get_next());
55                 cur->set_next(temp);
56                 count++;
57             }
58             return true ;
59         }
60         else
61             return false;
62     }
63     void InsertEnd(Type item)
64     {
65         Node<Type>* temp;
66         temp = new Node<Type>;
67         temp->set_item(item);
68         if (IsEmpty())
69             First = temp;
70         else
71             End->set_next(temp);
72         End = temp;
73         count++;
74     }
75     Type getFirst()
76     {
77         if (!IsEmpty())
78             return First->get_item();
```

```

79         else
80             return NULL;
81     }
82
83     Type getEnd()
84     {
85         if (!IsEmpty())
86             return End->get_item();
87         else
88             return NULL;
89     }
90
91     bool IsEmpty()
92     {
93         return !count;
94     }
95
96     int GetCount()
97     {
98         return count;
99     }
100     bool Find_Remove(Type val, Type& x)
101     {
102         Node<Type>* prev = NULL;
103         Node<Type>* ptr = First;
104         while (ptr)
105         {
106             if (*(ptr->get_item()) == val)
107             {
108                 if (prev == NULL)
109                 {
110                     x = ptr->get_item();
111                     First = First->get_next();
112                 }
113                 else
114                 {
115                     x = ptr->get_item();
116                     prev->set_next(ptr->get_next());
117                 }
118                 count--;
119                 return true;
120             }
121             prev = ptr;
122             ptr = ptr->get_next();
123         }
124         return false;
125     }
126
127     bool removeFirst(Type& x)
128     {
129         if (count == 0)
130             return false;
131
132         Node<Type>* ptr;
133         ptr = First;
134         if (count == 1)
135         {
136             First = NULL;
137             End = NULL;
138         }
139         else
140         {
141             First = First->get_next();
142         }
143         x = ptr->get_item();
144         count--;
145         return true;
146     }
147     void print()
148     {
149         Node<Type>* temp = First;
150         while (temp)
151         {
152             cout << (temp->get_item());
153             temp = temp->get_next();
154             if (temp)
155                 cout << ' ';

```

```
156         }  
157     }  
158  
159 };
```



# Main.cpp File Reference

Main of Project.

---

## Detailed Description

Main of Project.

## Version

0.1

## Copyright

Copyright secured by YMY Team(c) 2022

# 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

```
6 #pragma once
7 template<class Type>
8 class Node
9 {
10 private:
11     Type item;
12     int Priority;
13     Node<Type>* next;
14 public:
15     Node() :next(nullptr)
16     {
17     }
18     Node(const Type anitem)
19     {
20         item = anitem;
21         next = nullptr;
22     }
23     Node(const Type anitem, Node<Type>* nextptr)
24     {
25         item = anitem;
26         next = nextptr;
27     }
28     void set_item(const Type anitem) { item = anitem; }
29     void set_next(Node<Type>* nextptr) { next = nextptr; }
30
31     Type get_item()const
32     {
33         return item;
34     }
35     Node<Type>* get_next()const { return next; }
36
37
38 };
```

# 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.1

```
7 #pragma once
8 #include "Event.h"
9 #include "Time.h"
10 #include "Def.h"
11
12 class PreparationEvent :
13     public Event
14 {
15     CARGO_TYPE type;
16     float dist, loadTime, cost;
17 public:
18     PreparationEvent(Company* p, CARGO_TYPE, const Time&, int, float, float, float);
23     void Execute();
24 };
25
```

## PriNode.h

```
1 #pragma once
2 template<class Type>
3 class PriNode
4 {
5
6     private:
7         Type item;
8         int Priority;
9         PriNode<Type>* next;
10    public:
11        PriNode() :next(nullptr)
12        {
13            Priority = 0;
14        }
15        PriNode(const Type anitem)
16        {
17            item = anitem;
18            next = nullptr;
19            Priority = 0;
20        }
21        PriNode(const Type anitem, PriNode<Type>* nextptr)
22        {
23            item = anitem;
24            next = nextptr;
25            Priority = 0;
26        }
27        void set_item(const Type anitem) { item = anitem; }
28        void set_next(PriNode<Type>* nextptr) { next = nextptr; }
29        void set_priority(int p)
30        {
31            Priority = p;
32        }
33
34        Type get_item()const { return item; }
35        PriNode<Type>* get_next()const { return next; }
36        int get_priority()
37        {
38            return Priority;
39        }
40
41    };
42
43
44
```

# 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.[1](#)

```
7 #pragma once
8 #include "PriNode.h"
9 #include<iostream>
10
11 template<class Type>
12 class PriQueue
13 {
14
15 private:
16     PriNode<Type>* front;
17     int count;
18     PriNode<Type>* rear;
19 public:
20     PriQueue()
21     {
22         front = nullptr;
23         rear = nullptr;
24         count = 0;
25     }
26     bool EnQueue(Type item, float priority = 0)
27     {
28         PriNode<Type>* temp;
29         PriNode<Type>* cur = front;
30         temp = new PriNode<Type>;
31         temp->set_item(item);
32         temp->set_priority(priority);
33         if (QueueEmpty())
34         {
35             front = temp;
36             rear = temp;
37
38         }
39         else if (front->get_priority() < priority)
40         {
41             temp->set_next(front);
42             front = temp;
43         }
44         else
45         {
46             while (cur->get_next() != NULL && cur->get_next()->get_priority() >=
priority)
47                 cur = cur->get_next();
48             if (!cur->get_next())
49                 rear = temp;
50             temp->set_next(cur->get_next());
51             cur->set_next(temp);
52
53         }
54
55         count++;
56         return true;
57     }
58
59     bool DeQueue(Type& x)
60     {
61         if (count == 0)
62             return false;
63         PriNode<Type>* delptr;
64         delptr = front;
65         if (count == 1)
66         {
67             rear = NULL;
68             front = NULL;
69         }
70         else
71         {
72             front = front->get_next();
73         }
74         delptr->set_next(nullptr);
75         x = delptr->get_item();
76         count--;
77         return true;
```



```

78     }
79     bool QueueEmpty()
80     {
81         return !count;
82     }
83     void DistroyQueue()
84     {
85         Type x;
86         int c = count;
87         for (int i = 0; i < c; i++)
88             DeQueue(x);
89     }
90     PriNode<Type>* GetFront()
91     {
92         return front;
93     }
94     PriNode<Type>* GetRear()
95     {
96         return rear;
97     }
98     int GetCount()
99     {
100         return count;
101     }
102     Type Peek()
103     {
104         if (!QueueEmpty())
105             return front->get_item();
106         else
107             return NULL;
108     }
109
110     ~PriQueue()
111     {
112         DistroyQueue();
113     }
114
115     void print()
116     {
117         PriNode<Type>* temp = front;
118         while (temp)
119         {
120             cout << (temp->get_item());
121             temp = temp->get_next();
122             if (temp)
123                 cout << ',';
124         }
125     }
126 };
127

```

# PromoteEvent.h File Reference

Children Class of **Event**.

## Data Structures

- class **PromoteEvent**
- 

## Detailed Description

Children Class of **Event**.

## Version

0.1

## Copyright

Copyright secured by YMY Team(c) 2022

## PromoteEvent.h

Go to the documentation of this file.1

```
8 #pragma once
9 #include "Event.h"
10 class PromoteEvent :
11     public Event
12 {
13     //money used to promote the cargo, added to its original cost
14     float ExtraMoney;
15 public:
16     PromoteEvent(Company* p, const Time&, int, float);
17     //promotes a Normal cargo to VIP given its ID
18     void Execute();
19 };
20
```

## Queue.h

```
1 #pragma once
2 #include "Node.h"
3 #include <iostream>
4 using namespace std;
5
6 template<class Type>
7 class Queue
8 {
9 private:
10     Node<Type>* front;
11     int count;
12     Node<Type>* rear;
13 public:
14     Queue()
15     {
16         front = nullptr;
17         rear = nullptr;
18         count = 0;
19     }
20     bool EnQueue(Type item)
21     {
22         Node<Type>* temp;
23         temp = new Node<Type>;
24         temp->set_item(item);
25         if (QueueEmpty())
26             front = temp;
27         else
28             rear->set_next(temp);
29
30         rear = temp;
31         count++;
32         return true;
33     }
34
35     bool DeQueue(Type& x)
36     {
37         if (count == 0)
38             return false;
39         Node<Type>* delptr;
40         delptr = front;
41         if (count == 1)
42         {
43             rear = NULL;
44             front = NULL;
45         }
46         else
47         {
48             front = front->get_next();
49         }
50         delptr->set_next(nullptr);
51         x = delptr->get_item();
52         count--;
53         return true;
54     }
55     bool QueueEmpty()
56     {
57         return !count;
58     }
59     void DestroyQueue()
60     {
61         Type x;
62         int c = count;
63         for (int i = 0; i < c; i++)
64             DeQueue(x);
65     }
66     Node<Type>* GetFront()
67     {
68         return front;
69     }
70     Node<Type>* GetRear()
71     {
72         return rear;
73     }
```

```

74     int GetCount()
75     {
76         return count;
77     }
78     Type Peek()
79     {
80         if (!QueueEmpty())
81             return front->get_item();
82         else
83             return NULL;
84     }
85
86     ~Queue()
87     {
88         DestroyQueue();
89     }
90
91     void print()
92     {
93         Node<Type>* temp = front;
94         while (temp)
95         {
96             cout << temp->get_item();
97             temp = temp->get_next();
98             if (temp)
99                 cout << ',';
100         }
101     }
102 };
103

```

## Stack.h

```
1 #include "Node.h"
2 #include <cassert>
3
4 template <class T>
5 class Stack
6 {
7     Node<T>* Head;
8
9     void copyStack(const Stack<T>& R) {
10         if (R.Head == NULL)
11         {
12             IntializeStack();
13             return;
14         }
15         if (R.Head == Head)
16             return;
17         IntializeStack();
18         Node<T>* ptr;
19         Node<T>* Rptr;
20         Head = new Node<T>;
21         Head->setitem(R.Head->getitem());
22         ptr = Head;
23         Rptr = R.Head->getnext();
24         while (Rptr) {
25             ptr->getnext() = new Node<T>;
26             ptr = ptr->getnext();
27             ptr->setitem(Rptr->getitem());
28             Rptr = Rptr->getnext();
29         }
30         ptr->setnext(NULL);
31     }
32
33 public:
34     Stack() {
35         Head = NULL;
36     }
37     Stack(const Stack<T>& R) {
38         Head = NULL;
39         copyStack(R);
40     }
41     void IntializeStack() {
42         if (Head == NULL)
43             return;
44         Node<T>* ptr = Head;
45         Node<T>* nextptr;
46
47         while (ptr) {
48             nextptr = ptr->getnext();
49             delete ptr;
50             ptr = nextptr;
51         }
52         Head = NULL;
53     }
54     bool IsEmptyStack() {
55         return (Head == NULL);
56     }
57
58     const Stack<T>& operator =(const Stack<T>& R) {
59         copyStack();
60         return *this;
61     }
62
63     void Push(T data) {
64         Node <T>* ptr = new Node<T>(data);
65         if (Head == NULL)
66         {
67             Head = ptr;
68             Head->setnext(NULL);
69             return;
70         }
71     }
72
73 }
```

```
74         ptr->setnext(Head);
75         Head = ptr;
76     }
77
78     bool Pop() {
79         if (IsEmptyStack())
80             return false;
81         Node <T>* ptr = Head;
82         Head = Head->getnext();
83         delete ptr;
84         return true;
85     }
86     const T& Top() {
87         assert(Head != NULL);
88         return Head->getitem();
89     }
90     ~Stack() {
91         IntializeStack();
92     }
93 };
94
```

## 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.1

```
7 #pragma once
8 #include <string>
9 #include "UI.h"
10 using namespace std;
11
12 class Time
13 {
14     UI UI_P;
15     int hour;
16     int day;
17 public:
18     Time();
19     Time(int d , int h);
20     Time(string x);
21     Time(int h);
22     void setTime(int h);
23     void setTime(string x);
24     void AdvanceTime(int value);
25     void printTime();
26     int getHour();
27     int getDay();
28     int Time_In_Hours(); //returns (24*day+hour) which is the total number of hours
29     string Time_to_print();
30     bool operator==(const Time&);
31     bool operator>=(const Time&);
32     int operator-(const Time&);
33     Time operator +(float x);
34     ~Time();
35 };
36
```

# 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

<i>ostream&amp;</i>	out
<i>Truck*</i>	c

### Returns

ostream&

## Truck.h

Go to the documentation of this file.

```
7 #pragma once
8 #include<string>
9 #include"Def.h"
10 #include "Cargo.h"
11 #include "PriQueue.h"
12
13 using namespace std;
14 class Truck
15 {
16 private:
17
18     UI* ui_p;
19     PriQueue<Cargo*>container;
20     TRUCK_TYPE Type;
25     int Truck_Capacity;
30     float Maintenance_Time;
31     float Speed; //Km/h
36     int J;
41     int Journeys_Till_Check;
46     float Delivery_Interval;
51     float Delivery_Distance;
56     Time Nearest_stop;
61     float Nearest_dis;
62     int ID;
67     int move_counter;
72     Time finish_point;
77     Time AT;
82     int TDC;
87     int N;
88     Time moving_time;
89
90 public:
101     Truck(int id, TRUCK_TYPE T, int TC, float MT, int j, float S);
107     TRUCK_TYPE GetType() const;
113     int GetCapacity() const;
118     void set_finish_point(const Time&);
124     Time get_finish_point();
130     float GetMaintenanceTime() const;
136     float Get_nearest_dis();
142     float GetSpeed() const;
148     float GetDeliveryInterval();
153     void set_nearest_stop(Time, float);
159     Time Get_nearest_stop();
165     int GetJTC();
171     int GetContainer_count();
176     void restore_JTC();
181     void DecrementJTC();
187     int GetID() const;
192     void set_DInterval();
198     void load(Cargo*, float delivery_time);
203     Cargo* unload();
204     void count_down();
205     int get_move_counter();
206     void Set_AT(int h);
207     void inc_TDC();
208     void inc_N();
209     int Get_TDC();
210     int Get_N();
211     Time Get_AT();
218     float utilization(Time& Sim Time);
219     void print_container();
220     void print();
221     void Set_moving_time(Time& Sim_Time);
222     Time get_moving_time();
223
224 };
232 ostream& operator<<(ostream& out, Truck* t);
```

# UI.h File Reference

**Truck** Class.

## Data Structures

- class **UI**

---

## Detailed Description

**Truck** Class.

## Version

0.1

## Copyright

Copyright secured by YMY Team(c) 2022

## UI.h

Go to the documentation of this file.

```
1 #pragma once
2 #include "Def.h"
3 #include <iostream>
4 #include <string>
5 using namespace std;
6
7 class UI
8 {
9
10 public:
11     UI();
12     void print(string);
13     int getInteger();
14     string getString();
15 };
16
17
```

