Ministry of Science and Education of Russian Federation
Peter the Great St. Petersburg Polytechnic University
—
Institute of Applied Mathematics and Mechanics
Higher school of cybersecurity and information security

L A B № 1

«Creation of UML diagram»

course «OOP»

Student
Gr.3651003/70801 Gasanov E.A.

<signature>

Instructor
Chernov A.Y.

<signature>

Saint-Petersburg

2019

Choosing and describing subject area

According to the given task, we are to choose the subject area all by ourselves. In this case I've made a choice to describe using UML language a passenger who's departing from local airport. All the details of are represented in the table below.

Entity type	*Class Passenger*
The purpose of the type	*Represents passengers who is departing from local airport*
Attributes	*+ Passenger: Human // a regular human*
Operations	*+Hand_luggage()* *+LookATScreen(Passenger:Human,info:info_board)* *+ DoAction2(deal: Deal*,money:unsighned int): void virtual* *+ForeignPassport()* *+Boarding_ticket()* *+DebitCard()*

Entity type	*Interface Person_Security*
The purpose of the type	*Represents a collection of functions(abstract class) which have only declaration to check a passenger and their stuff but no the implementation*
Attributes	
Operations	*+ ~Person_security():virtual* *+ DoAction(human_to_check: Human, any_luggage:luggage_mass *): virtual void=0*

Entity type	*Class Security on entrance*
The purpose of the type	*Represents a security who are to check passengers' luggage and their hand luggage*
Attributes	*- metal_detector : MetalD // metal detector* *- x_ray_luggage : XrayLugg // x-ray for luggage*
Operations	*+ Security on entrance() // constructor* *+DoAction(human_to_check: Human, any_luggage:luggage_mass *): virtual void*

Entity type	*Class luggage common*
The purpose of the type	*Represents passenger's luggage*
Attributes	*+ C_Luagge: luggage_mass * // the struct of luggage(luggage and hand-luggage)* *+amount : unsigned int // amount of common luggage of 1 passager*
Operations	*+ Be_moved():void*

Entity type	*Class Big_luggage*
The purpose of the type	*Represents heavy luggage*
Attributes	
Operations	*+Big_Luggage()*

Entity type	*Class Luggage_claim*
The purpose of the type	*Represents a reception desk where heavy luggage is given*
Attributes	*# weight : unsigned int // weight of hleavy luggage* *# Heavy_luggage: luggage_mass_heavy* // mass of heavy luggage*
Operations	*#TakeHeavy(big_luggage:luggage_mass_heavy*):void* *-WeightLuggage(big_luggage:luggage_mass_heavy*) :void* *+Take_pass(passport:f_passport): void* *+Give_pass(passport:f_passport):void*

Entity type	*Class Foreign_passport*
The purpose of the type	*Represents a foreign_passport*
Attributes	*+passport:f_passport*
Operations	*+Give_pass(passport:f_passport)* *+Take_pass(passport:f_passport)*

Entity type	*Class Hand_luggage*
The purpose of the type	*Represents a hand_luggage*
Attributes	*+ hand_lug : luggage_mass* //*
Operations	

Entity type	*Class Personal_inspection*
The purpose of the type	*Represents inspection of passengers and their stuff*
Attributes	*-x_ray_hum : xrhum // xray hum* *-xray_hand_lug: xrlug*
Operations	*+Personal_inspectation() // constructor* *+DoAction(human_to_check: Human, any_luggage:luggage_mass *): virtual void*

Entity type	*Class Boarding_ticket*
The purpose of the type	*Represents a ticket to board a plane*
Attributes	*+board_ticket : b_ticket // Boarding ticket*
Operations	*+ Show(board_ticket : b_ticket): void*

Entity type	*Class DebitCard*
The purpose of the type	*Represents passenger's money*
Attributes	*- balance : unsigned int*
Operations	*+ Pay() : void*

Entity type	*Class Waiting room*
The purpose of the type	*Represents a place where passengers are waiting for a flight*
Attributes	*-bench: Bench* // a sit down stuff* *-power_socket:socket*
Operations	*+ Waiting_room(): void // constructor*

Entity type	*Class Bench*
The purpose of the type	*Represents a bench*
Attributes	*-length: unsigned int* *-amount: unsigned int // the num of benches*
Operations	*+Bench()*

Entity type	*Class infoboard*
The purpose of the type	*Represents a board where the info about flights is displayed*
Attributes	*Info:info_board // info about gate,flight…*
Operations	*- Refresh():void* *+ infoboard()*

Entity type	*Class Airport*
The purpose of the type	*Represents a place where passengers are getting abroad*
Attributes	*-Hangar: hangar** *-Mechanics : mechanics** *-Telescopic_ladder:t_ ladder** *-Special_transport: s_transport** *-Emergency_transport: e_transport** *-Dispetchers: dispetcher** *+Custom_officer:c_officer**
Operations	*+ Passport_control(passport: f_passport):void* *-Ground_services(Special_transport: s_transport*): void* *-Navigation(Dispetchers: dispetcher*):void* *-Luggage_delivery(Special_transport: s_transport*):void* *-Repair(Mechanics:machanics*,Hangar:hangar*):void* *-Boarding(Passenger: Human, Telescopic_ladder:t_ ladder*):void* *-Airport()*

Entity type	*Class Desires*
The purpose of the type	*Wishes of passenger in the airport*
Attributes	*-McDonalds: fast_food* *-DutyFree: shop* *-Starbucks: c_shop* *-Restroom: lavatory*
Operations	*- eat_drink(Passenger: Human,boarding_ticket:b_ticket) :void* *-shopping(Passenger: Human,boarding_ticket:b_ticket): void* *+Pay():void* *-washing(Passenger: Human):void* *-go_to_waitingroom(Passenger: Human):void*

Description of the C project

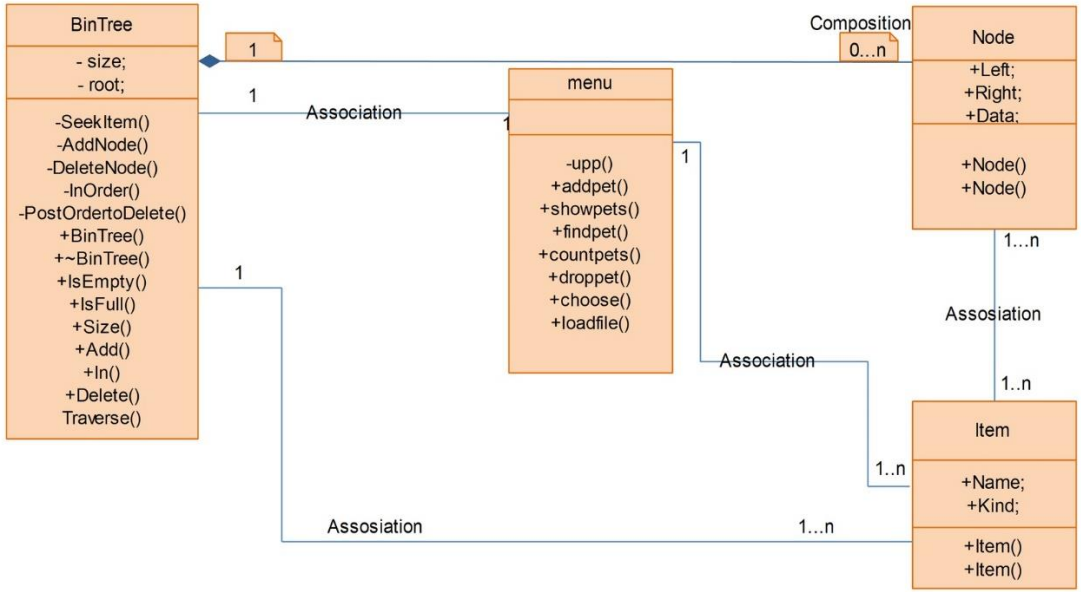
We are to choose any C project to be transformed to the C++ language. My choice is – binary search tree. A binary tree is either an empty set of nodes (an empty tree) or a set of nodes with one node designated the root. Each node has exactly two trees, called the left subtree and the right subtree, descending from it. Each subtree is itself a binary tree, which includes the possibility of being an empty tree. A binary search tree is an ordered binary tree in which each node contains an item, in which all items in the left subtree precede the root item, and in which the root item precedes all items in the right subtree. The developed data structure is representing the roster of the Nerfville Pet Club, with each item consisting of a pet name and a pet kind. Let's name types of operations: determining whether tree is empty, determining whether tree is full, determining the number of items in the tree, adding an item to the tree, removing an item from the tree, searching the tree for an item, visiting each item in the tree, loading from file. The table below has a description of each class and its methods.

Entity type	*Class BinTree*
The purpose of the type	*Represents an ADT of binary search tree*
Attributes	*-size: int //size of tree* *-root: Node**
Operations	*-SeekItem(data:Item):pair<Node*, Node*>* *- AddNode(cur:Node*&,new_node:Node*):void* *- DeleteNode(cur:Node*&):void* *- InOrder(obj:Node*):void* *- PostOrdertoDelete(obj:Node*):void* *+BinTree()* *+~BinTree()* *+IsEmpty():bool* *+ IsFull():bool* *+Size(:const): const* *+Add(data:Item):bool* *+In(data:Item):bool* *+Delete(data:Item):bool* *+ Traverse():void*

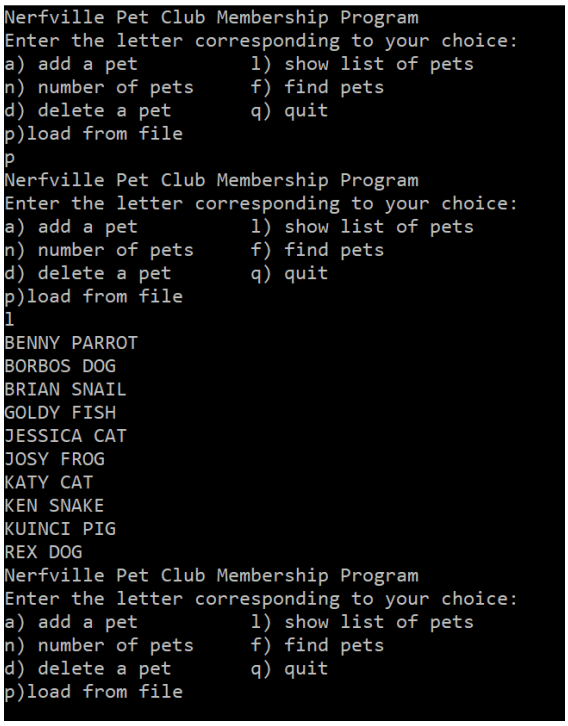
Entity type	*Class menu*
The purpose of the type	*Represents a collection of static functions so it's not necessary to create an object*
Attributes	
Operations	*+addpet(mytree:BinTree&):static void* *+ showpets (mytree:BinTree&):static void* *+ findpet (mytree:BinTree&):static void* *+ countpets (mytree:BinTree&):static void* *+ droppet (mytree:BinTree&):static void* *+ choose ():static char* *+ loadfile (mytree:BinTree&):static void* *+ upp (str:string&):static void*

Entity type	*Class Item*
The purpose of the type	*Represents the information of pets*
Attributes	*+Name: string* *+Kind : string*
Operations	*+Item()* *+Item(name:string &,kind:string&)*

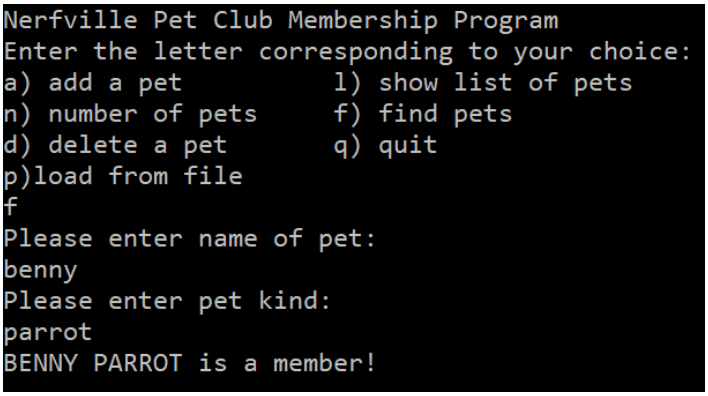
Entity type	*Class Node*
The purpose of the type	*Represents each node of binary search tree*
Attributes	*+Left:Node** *+Right:Node** *+Data:Item*
Operations	*+Node(data:item,left:Node*,right:Node*)* *+Node(source:const Node&)*


BinTree
- size;
- root;
-SeekItem()
-AddNode()
-DeleteNode()
-InOrder()
-PostOrdertoDelete()
+BinTree()
+~BinTree()
+IsEmpty()
+IsFull()
+Size()
+Add()
+In()
+Delete()
Traverse()
1
Composition
0...n
Node
+Left;
+Right;
+Data;
+Node()
+Node()
1
Association
1
menu
-upp()
+addpet()
+showpets()
+findpet()
+countpets()
+droppet()
+choose()
+loadfile()
1
1...n
Assosiation
1..n
Association
1
1..n
Item
+Name;
+Kind;
+Item()
+Item()
Assosiation
1...n

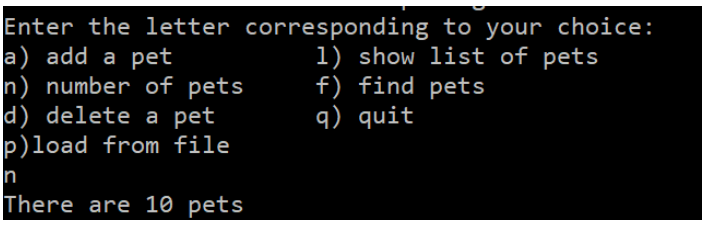
Screenshots of developed program



Pic.1, loading from file and listing pets



Pic.2, finding pets



Pic.3, amount of pets

Source code of developed program

BinTree.h

```
#pragma once
#include "iostream"
#include "item.h"
#include "Node.h"
#include"string"
#define MAXITEMS 10

using namespace std;

class BinTree
{
public:
	BinTree();

	~BinTree();

	bool IsEmpty();
	bool IsFull();
	const int Size() const;

	bool Add(Item data);
	bool In(Item data);
	bool Delete(Item data);
	void Traverse();

private:
	int size;
	Node* root;

	pair<Node*, Node*> SeekItem(Item data);
	void AddNode(Node *&cur, Node* new_node);
	void DeleteNode(Node*& cur);
	void InOrder(Node* obj);
	void PostOrdertoDelete(Node* obj);

};
```

BinTree.cpp

```
#include "BinTree.h"

BinTree::BinTree()
{
	root = NULL;
	size = 0;
}

BinTree::~BinTree()
{
	/*if (root != NULL)
	{
		InOrder(root->Left);
		InOrder(root->Right);
		root->~Node();
	}
	root = NULL;
	size = 0;*/
	if (root != NULL)
```

```
	{
		PostOrdertoDelete(root);
	}
	//delete root;
	root = NULL;
	size =0;
}

bool BinTree::IsEmpty()
{
	return root == NULL;
}

bool BinTree::IsFull()
{
	return size == MAXITEMS;
}

const int BinTree::Size() const
{
	return size;
}

bool BinTree::Add(Item data)
{
	Node* new_node;

	if (IsFull())
	{
		cerr << "Tree is full" << endl;
		return false;
	}
	if (SeekItem(data).second)
	{
		cerr << "Attempted to add duplicate item" << endl;
		return false;
	}

	new_node = new Node(data);

	if (!new_node)
	{
		cerr << "Couldn't create node" << endl;
		return false;
	}

	++size;

	if (!root)
		root = new_node;
	else
		AddNode(root, new_node);
	return true;
}

bool BinTree::In(Item data)
{
	return SeekItem(data).second;
}

bool BinTree::Delete(Item data)
{
	pair<Node*, Node*> look = SeekItem(data);
	if (!look.second)
	{
```

```
                return false;
        }
        else if (!look.first)
        {
                DeleteNode(root);
        }
        else if (look.first->Left == look.second)
        {
                DeleteNode(look.first->Left);
        }
        else
        {
                DeleteNode(look.first->Right);
        }
        size--;
        return 1;
}

void BinTree::Traverse()
{
        InOrder(root);
}

pair<Node*, Node*> BinTree::SeekItem(Item data)
{
        //pair<Node*, Node*> look = make_pair((Node*)NULL, root);
        //look.first - parent
        //look.second - child
        pair <Node*, Node*> look((Node*)NULL, root);
        if (!look.second)
                return look;

        for (; look.second;)
        {
                if (look.second->Data > data)
                {
                        look.first = look.second;
                        look.second = look.first->Left;
                }
                else if (look.second->Data < data)
                {
                        look.first = look.second;
                        look.second = look.first->Right;
                }
                else
                        break;
        }
        return look;
}

void BinTree::AddNode(Node *&cur, Node* new_node)
{
        if (!cur)
        {
                cur = new_node;
                return;
        }
        if (cur->Data > new_node->Data)
                AddNode(cur->Left, new_node);
        else
                AddNode(cur->Right, new_node);
}

void BinTree::DeleteNode(Node*& cur)
{
```

```
	Node* temp = cur;
	if (!cur->Left)
	{
		cur = cur->Right;
	}
	else if (!cur->Right)
	{
		cur = cur->Left;
	}
	else
	{
		for (temp = cur->Left; temp->Right; temp = temp->Right);
		temp->Right = cur->Right;
		temp = cur;
		cur = cur->Left;
	}
	delete temp;
}

void BinTree::InOrder(Node* obj)
{
	if (obj)
	{
		InOrder(obj->Left);
		cout << obj->Data.Name<<" ";
		cout << obj->Data.Kind << endl;
		InOrder(obj->Right);
	}
}
void BinTree::PostOrdertoDelete(Node* obj)
{
	//Node* pright;
	if (obj)
	{
		//pright = obj->Right;
		PostOrdertoDelete(obj->Left);
		PostOrdertoDelete(obj->Right);
		delete obj;
	}
}
```

Menu.h

```
#pragma once
#define _CRT_SECURE_NO_WARNINGS
#include "BinTree.h"
#include <fstream>
#include <algorithm>

class menu
{
private:

	static void upp(string& str);
public:

	static void addpet(BinTree &mytree);
	static void showpets(BinTree &mytree);
	static void findpet(BinTree& mytree);
	static void countpets(BinTree& mytree);
	static void droppet(BinTree& mytree);
	static char choose();
	static void loadfile(BinTree& mytree);
};
void menu::upp(string& str)
```

```
{
	unsigned k = str.length();
	//std::transform(str.begin(), str.end(), str.begin(), ::toupper);
	for (unsigned i = 0;i< k; i++)
	{
		str[i] = toupper(str[i]);
	}

}
void menu::addpet(BinTree &mytree)
{
	Item temp;
	cout << "Please enter name of pet: " << endl;
	//std:sin >> temp.Name;
	cin.ignore(32767, '\n');
	getline(std::cin, temp.Name);
	cout << "Please enter pet kind: " << endl;
	getline(std::cin, temp.Kind);
	upp(temp.Name);
	upp(temp.Kind);
	mytree.Add(temp);
}
void menu::showpets(BinTree &mytree)
{
	if (mytree.IsEmpty())
	{
		cerr << "No entries!";
	}
	else
		mytree.Traverse();

}
void menu::findpet(BinTree& mytree)
{

	Item temp;
	if (mytree.IsEmpty())
	{
		cerr << "No entries!";
		return;
	}

	cout << "Please enter name of pet: " << endl;
	//std:sin >> temp.Name;
	cin.ignore(32767, '\n');
	getline(std::cin, temp.Name);
	cout << "Please enter pet kind: " << endl;
	getline(std::cin, temp.Kind);
	upp(temp.Name);
	upp(temp.Kind);
	cout << temp.Name <<" "<< temp.Kind<<" ";
	if (mytree.In(temp))
	{
		cout << "is a member!" << endl;
	}
	else
		cout << "is not a member!" << endl;

}
void menu::countpets(BinTree& mytree)
{
	cout << "There are "<<mytree.Size()<<" pets"<<endl;
}
void menu::droppet(BinTree& mytree)
{
```

```
        Item temp;
        if (mytree.IsEmpty())
        {
                cerr << "No entries!";
                return;
        }

        cout << "Please enter name of pet you wish to delete: " << endl;
        //std:sin >> temp.Name;
        cin.ignore(32767, '\n');
        getline(std::cin, temp.Name);
        cout << "Please enter pet kind: " << endl;
        getline(std::cin, temp.Kind);
        upp(temp.Name);
        upp(temp.Kind);
        cout << temp.Name << " "<< temp.Kind;
        if (mytree.Delete(temp))
        {
                cout << " is deleted from the club"<<endl;
        }
        else
                cout << " is not a member"<< endl;


}

char menu::choose()
{
        char ch;
        cout << "Nerfville Pet Club Membership Program"<<endl;
        cout << "Enter the letter corresponding to your choice:" << endl;
        cout << "a) add a pet          l) show list of pets" << endl;
        cout << "n) number of pets     f) find pets" << endl;
        cout << "d) delete a pet       q) quit" << endl;
        cout << "p)load from file" << endl;

        string letters = "alrfndq";
        while (1)
        {
                cin >> ch;
                //cin.ignore(32767, '\n');
                ch = tolower(ch);
                if (!(letters.find_first_of(ch, 1)))
                {
                        cerr << "Please enter an a,l,n,f,d,q";
                }
                else
                        break;
        }
        if (ch == EOF)
                ch = 'q';

        return ch;
}
```

Results

We have understood how to deal with basic UML operations. Tried all by ourselves to create the UML diagram. Translated the C code into C++ using OOP.