152113017: NESNE TABANLI PROGRAMLAMA I
FINAL SINAVI (CEVAPLAR)
Toplam: 100 puan, 3 Soru, Süre: 300 dakika

S.1	S.2	S.3	Toplam
30	30	40	100

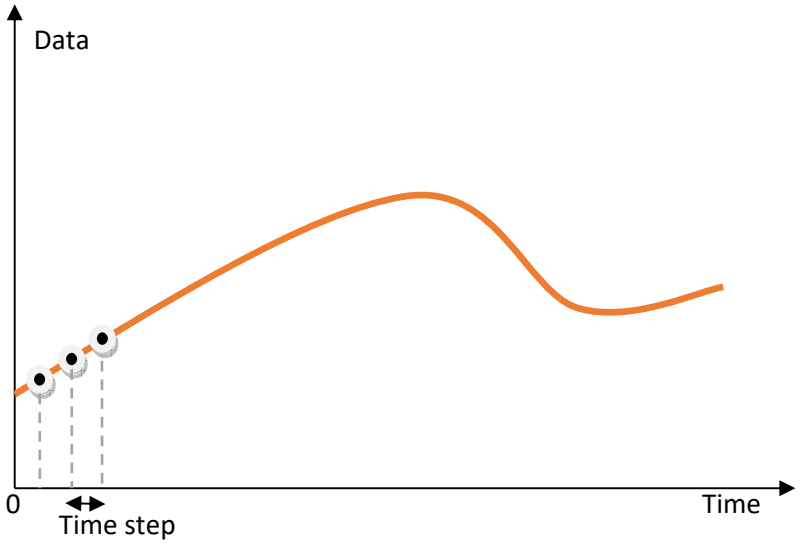
Adı-Soyadı: CEVAPLAR	Numara: CEVAPLAR

S.1 Sizden bir `Graph` adlı şablon (template) sınıf yazmanız istenmektedir. Bu sınıfın iki üyesi olacaktır. Bunlardan birisi `timeStep` olacaktır. Diğeri ise, std::vector taşıyıcısında tutulan `data`lar olacaktır. Böylece, bir datanın vektörde bulunduğu indeks timeStep ile çarpıldığında grafikte datanın zamanını bulmuş olursunuz.

- timeStep ve datalar farklı tiplerde olabilmektedir. Örneğin, timeStep double iken, datalar float olabilir. timeStep int iken datalar double olabilir vb.
- Kurucu fonksiyonda, parametre olarak timeStep değeri verilmektedir.
- `addValue` adlı üye fonksiyonu, parametre olarak verilen değeri bir sonraki timeStep için grafiğe data eklemesini yapmaktadır.
- `getMax` üye fonksiyonu, grafikteki maksimum değeri döndürmektedir.
- `getMin` üye fonksiyonu, grafikteki minimum değeri döndürmektedir.
- += üye operatör fonksiyonu, addValue gibi işlem yapmaktadır. Örneğin, Graph graph; graph+=5; yazdığınızda, 5 değeri yeni bir değer olarak eklenmektedir.
- Veri saklama, ara yüzü icradan ayırma <Seperating Interface from Implementation> olarak derste ifade edilen prensibe uygun yazılmalıdır.

(a) [20p] Sınıfı kodlayınız.

(b) [10p] Bir main programı yazınız. timeStep (float) ve data (double) olacak şekilde bir Graph nesnesi yaratınız. 2 * cos(PI / 4 * t + PI / 3), $0 \leq t \leq 5$ fonksiyonunun 0.1 zaman adımları için Graph nesnesine atınız. Burada hem operatör hem de addValue fonksiyonlarının kullanımı gereklidir. Sonra, getMin ve getMax fonksiyonlarını çağırarak test ediniz.



Cevap:
(a)

```
#include <iostream>
#include <vector>
using namespace std;

#define PI 3.1416

template<typename T, typename D>
class Graph {
private:
	vector<D> value;
	T timeStep;
public:
	Graph(T tStep);
	void addValue(D& data);
	D getMax() const;
	D getMin() const;
	void operator+=(D& data);
};
```

```
template<typename T, typename D>
Graph<T, D>::Graph(T tStep)
	:timeStep(tStep)
{
}

template<typename T, typename D>
void Graph<T, D>::addValue(D& data)
{
	value.push_back(data);
}

template<typename T, typename D>
D Graph<T, D>::getMax() const
{
	D tmp;
	if(value.size()>0)
		tmp = value[0];
	for (int i = 0; i < value.size(); i++) {
		if(tmp<value[i]){
			tmp = value[i];
		}
	}
	return tmp;
}

template<typename T, typename D>
D Graph<T, D>::getMin() const
{
	D tmp;
	if (value.size() > 0)
		tmp = value[0];
	for (int i = 0; i < value.size(); i++) {
		if (tmp > value[i]) {
			tmp = value[i];
		}
	}
	return tmp;
}

template<typename T, typename D>
void Graph<T, D>::operator+=(D& data)
{
	value.push_back(data);
}
```

(b)

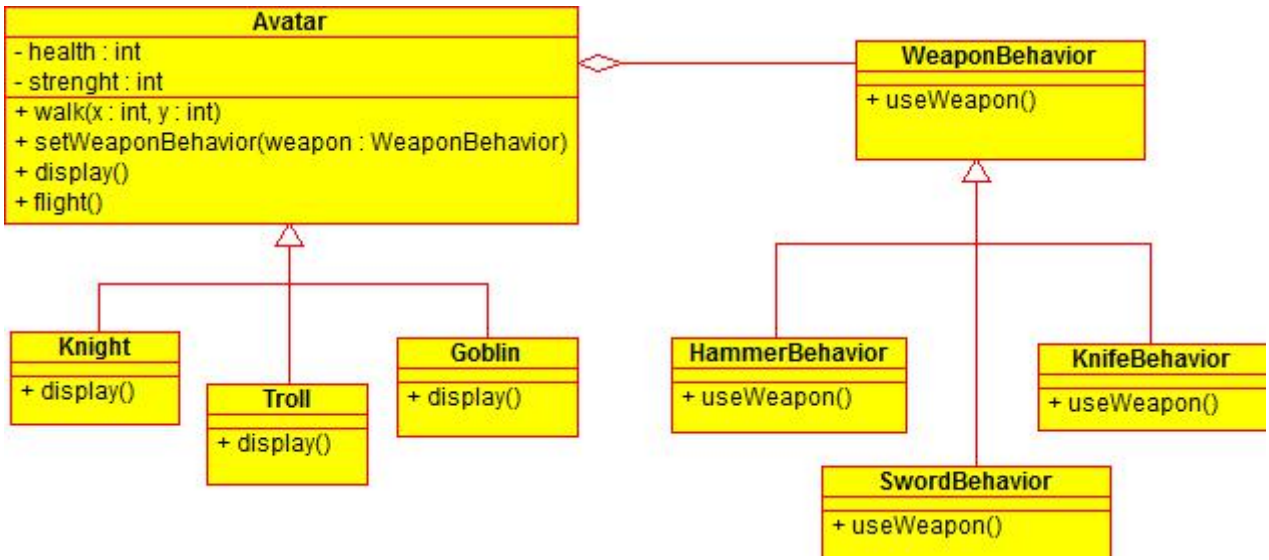
```
int main() {
	Graph<float, double> graph(0.1);
	double value;

	for (float t = 0; t < 5; t = t + 0.1) {
		value = (2 * cos(PI / 4 * t + PI / 3));
		graph.addValue(value);
	}
	graph += value;
	cout << "Graph max value is " << graph.getMax() << endl;
	cout << "Graph min value is " << graph.getMin() << endl;

	return 0;
}
```

S.2 Bir PC oyunun ilk versiyonunda, "Knight", "Goblin" ve "Troll" adlı avatarlar vardır (İlerde yeni karakterler eklenme durumu vardır). Bütün avatarlar, "walk" metodu ile yürümektedir. Her avatarın, "health" değeri vardır. Bu sağlık durumunu göstermektedir. Ayrıca, "strength" gücünü ifade etmektedir. Her Avatar "display" olarak adlandırılan ve ekranda görülmesini sağlayan metoda sahiptir. İlk versiyonda, üç çeşit alet vardır ve bunlar hammer, sword, ve knife olarak adlandırılmaktadır (Belki sonraki versiyonda yenileri eklenebilir). Her aletin kullanımı farklı sonuçlar doğurmaktadır. Oyunda, avatarlar kullandıkları aletleri değiştirebilmektedir. Bundan dolayı, aletleri useWeapon ortak fonksiyonu ile kullanmak gerekmektedir.

- Avatar ve WeaponBehavior, `abstract` sınıflardır.
- WeaponBehavior sınıfının useWeapon üye fonksiyonu pure virtual fonksiyondur.
- Avatarın display fonksiyonu pure virtual fonksiyondur.
- HammerBehavior sınıfının useWeapon fonksiyonu sadece "Hammer::Weapon" ekrana yazdırır. Benzer şekilde, Sword Behavior sınıfının useWeapon fonksiyonu Sword::Weapon" yazdırır. Diğeri de benzer şekilde.
- Knight sınıfının, display fonksiyonu "Knight::display" yazdırır. Diğerleri de benzer şekilde.
- flight fonksiyonu hangi alet setWeaponBehavior fonksiyonu ile Avatara verildi ise, o aletin useWeapon fonksiyonunu çağırır.
- Health ve straight, değişkenlerine rastgele değer atayabilirsiniz. Bunlarla pek işimiz yok. Sadece senaryoyu tamamlıyorlar.
- Walk fonksiyonu sadece x ve y parametre değerlerini ekrana yazdırıyor.



a) [20p] Sınıfları kodlayınız. Tüm sınıfları tek bir kaynak dosyada yazın. Fonksiyon kodlamalarını direk sınıf tanımı içinde verebilirsiniz. En hızlı nasıl yazabiliyorsanız o formda yazınız.

b) [10p] Bir uygulama yazınız. Knight avatarı oluşturup, önce bu avatara Hammer verin ve flight fonksiyonunun çağırın. Sonra aynı avatara Knife verin ve flight fonksiyonunu çağırın.

Cevap:

(a)

```
class WeaponBehavior {
public:
	virtual void useWeapon() = 0;
};
class HammerBehavior :public WeaponBehavior {
public:
	void useWeapon() {
		cout << "Hammer::Weapon" << endl;
	}
};
class KnifeBehavior :public WeaponBehavior {
public:
	void useWeapon() {
		cout << "Knife::Weapon" << endl;
	}
};
class SwordBehavior :public WeaponBehavior {
public:
	void useWeapon() {
		cout << "Sword::Weapon" << endl;
	}
};
class Avatar {
public:
	Avatar() {
		health = 0;
		strenght = 0;
	}
	virtual void display() = 0;
	void walk(int x, int y) {
		cout << "walking to (" << x << "," << y << ")" << endl;
	}
	void setWeaponBehavior(WeaponBehavior* w)
	{
		this->weapon = w;
	}
	void flight() { weapon->useWeapon(); }
private:
	WeaponBehavior* weapon;
	int health;
	int strenght;
};

class Knight :public Avatar {
public:
	void display() {
		cout << "Knight::display" << endl;
	}
};
class Troll :public Avatar {
public:
	void display() {
		cout << "Troll::display" << endl;
	}
};
class Goblin :public Avatar {
public:
	void display() {
		cout << "Goblin::display" << endl;
	}
};
```

(b)

```
int main() {
	//Creates a new avatar in the game
	Avatar* avatar = new Knight();

	avatar->setWeaponBehavior(new HammerBehavior());
	avatar->display();
	avatar->flight();
	avatar->setWeaponBehavior(new KnifeBehavior());
	avatar->display();
	avatar->flight();

	return 0;
}
```

S.3 `Ders`, `öğrenci` ve `kayıt` olmak üzere 3 sınıf oluşturmanız istenmektedir.

- `Ders`, kod, ad, kredi ve öğrencilerin tutulduğu bir `vektör` içermektedir.
- `Ders` kurucu fonksiyonunda, ders kod, ad ve kredisi atanmaktadır.
- `Ders`, dersi alan öğrencinin eklendiği `ogrenciEkle` üye fonksiyonuna sahiptir. Parametre olarak verilen öğrenci dersi alanlar arasına eklenmektedir.
- `Ders`, ders bilgisini (kod, ad, kredi) ekrana yazdıran `print` adlı üye fonksiyona sahiptir.
- `Ders`, dersi alan tüm öğrencilerin bilgilerini (No, ad, soyad) ekrana yazdıran `listele` adlı bir fonksiyona sahiptir.
- `Öğrenci`, no, ad, soyad ve aldığı derslerin tutulduğu bir `vektör` içermektedir.
- `Öğrenci`, kurucu fonksiyonunda, no, ad ve soyad atanmaktadır.
- `Öğrenci`, aldığı dersin eklendiği `dersEkle` üye fonksiyonuna sahiptir. Parametre olarak verilen ders öğrencinin aldığı dersler arasına eklenmektedir.
- `Öğrenci`, öğrenci bilgisini (no, ad, soyad) ekrana yazdıran `print` adlı üye fonksiyona sahiptir.
- `Öğrenci`, aldığı tüm ders bilgilerini (kod, ad, kredi) ekrana yazdıran `listele` adlı bir fonksiyona sahiptir.
- `Kayit`, yok edilmesinden sorumlu olduğu list olarak öğrenci ve dersler listesine sahiptir.
- `Kayit`, `yeniOgrenciKaydet` üye fonksiyonu ile listesine eğer yoksa parametre olarak verilen öğrenciyi eklemektedir.
- `Kayit`, `yeniDersKaydet` üye fonksiyonu ile listesine eğer yoksa parametre olarak verilen dersi eklemektedir.
- `Kayit`, `ogrenciDerseKaydet` üye fonksiyonu ile ilk parametrede öğrenci no, ikinci parametre olarak ders kodunu almaktadır. Eğer verilen noya sahip öğrenci listede varsa ve verilen koda sahip ders listed varsa, öğrencinin `dersEkle`, dersin `ogrenciEkle` fonksiyonun kullanarak ilişkilendirmeyi yapar.
- `Kayit`, `ogrenciListele` fonksiyonu ile listesinde bulunan öğrencilerin `print` fonksiyonlarını çağırmak suretiyle listeyi ekrana döker.
- `Kayit`, `dersListele` fonksiyonu ile listesinde bulunan derslerin print fonksiyonlarını çağırmak suretiyle listeyi ekrana döker.
- `Kayit`, `dersKayitlari` fonksiyonu ile parametrede verilen koda sahip derse kayıt olan öğrencilerin listesini ekrana yazdırır.
- `Kayit`, `kayitliDersler` fonksiyonu ile parametrede verilen numaraya sahip öğrencinin kayıt olduğu derslerin listesini ekrana yazdırır.
- Veri saklama, ara yüzü icradan ayırma <Seperating Interface from Implementation> olarak derste ifade edilen prensibe uygun yazılmalıdır.
- Ogrenci sınıfını kodlandığı .h dosyası başına "`class Ders;`" beyanını, Ders sınıfının kodlandığı .h dosyasının başına "`class Ogrenci;`" beyanını yazmayı unutmayın. Her iki sınıfta birbirlerine bağımlı olduğu için bunu yapmazsanız hata alabilirsiniz.

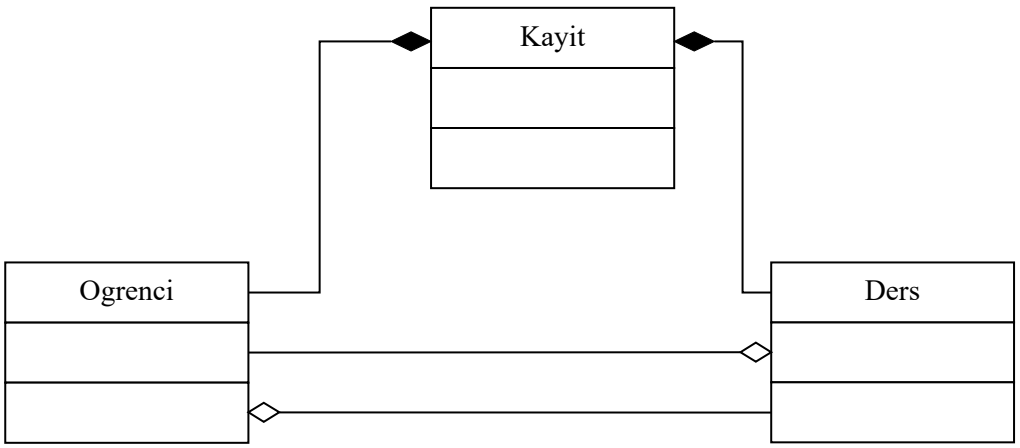
(a) [10p] UML diyagramını çiziniz (Sınıfların gösterimini yaparken, üyeleri vermeyin. Sadece sınıf isimlerini yazın. Bir de ilişkileri doğru şekilde gösterin. Composition ve aggregation gösterimleri önemli).

(b) [20p] Sınıfları kodlayınız.

(c) [10p]Bir Kayit nesnesi yaratın, iki ders ve 3 öğrenci ekleyin. Bir öğrenciyi bir ders diğer derse de diğer iki öğrenciyi kayıt ettirin. Listelemeleri yapın, herhangi bir ders dersKayitlari ve herhangi bir öğrenci için kayitliDersler çağırın.

Cevap:

(a)



(b)

Ders.h

```
class Ogrenci;

class Ders {
private:
	string Kod;
	string Ad;
	int Kredi;
	vector<Ogrenci *> ogrenciler;
public:
	Ders(string kod = "", string ad = "", int kredi = 0);
	bool ogrenciEkle(Ogrenci* ogr);
	string getKod() const;
	void print() const;
	void listele() const;
};
```

Ogrenci.h

```
class Ders;
class Ogrenci {
private:
	long No;
	string Ad;
	string Soyad;
	vector<Ders *> dersler;
public:
	Ogrenci(long no = 0, string ad = "", string soyad = "");
	long getNo() const;
	bool dersEkle(Ders* ders);
	void print() const;
	void listele() const;
};
```

Kayit.h

```
class Kayit {
private:
	list<Ogrenci*> ogrenciler;
	list<Ogrenci*>::iterator ogrIter;
	list<Ders*> dersler;
	list<Ders*>::iterator dersIter;
public:
	Kayit();
	~Kayit();
	bool yeniOgrenciKaydet(Ogrenci *ogr);
	bool yeniDersKaydet(Ders *ders);
	bool ogrenciDerseKaydet(long no, string kod);
	void ogrenciListele();
	void DersListele();
	void dersKayitlari(string kod);
	void kayitliDersler(long no);

};
```

Ders.cpp

```
Ders::Ders(string kod, string ad, int kredi)
	:Kod(kod), Ad(ad), Kredi(kredi)
{}
bool Ders::ogrenciEkle(Ogrenci* ogr) {
	vector<Ogrenci*>::iterator iter = find(ogrenciler.begin(), ogrenciler.end(), ogr);
	if (iter == ogrenciler.end()) {
		ogrenciler.push_back(ogr);
		return true;
	}
	else
		return false;
}

string Ders::getKod() const {
	return Kod;
}

void Ders::print() const {
	cout << Kod << ", " << Ad << ", " << Kredi << endl;
}

void Ders::listele() const {
	for (int i = 0; i < ogrenciler.size(); i++) {
		ogrenciler[i]->print();
		cout << endl;
	}
}
```

Ogrenci.cpp

```
Ogrenci::Ogrenci(long no, string ad, string soyad)
	:No(no), Ad(ad), Soyad(soyad)
{}

long Ogrenci::getNo() const {
	return No;
}

bool Ogrenci::dersEkle(Ders* ders) {
	vector<Ders*>::iterator iter = find(dersler.begin(), dersler.end(), ders);
	if (iter == dersler.end()) {
		dersler.push_back(ders);
		return true;
	}
	else
```

```
		return false;
}

void Ogrenci::print() const {
	cout << No << ", " << Ad << ", " << Soyad << endl;
}

void Ogrenci::listele() const {
	for (int i = 0; i < dersler.size(); i++) {
		dersler[i]->print();
		cout << endl;
	}
}
```

Kayit.cpp

```
Kayit::Kayit() {}

Kayit::~Kayit() {
	for (ogrIter = ogrenciler.begin(); ogrIter != ogrenciler.end(); ogrIter++) {
		delete (*ogrIter);
	}
	ogrenciler.clear();
	for (dersIter = dersler.begin(); dersIter != dersler.end(); dersIter++) {
		delete (*dersIter);
	}
	dersler.clear();
}

bool Kayit::yeniOgrenciKaydet(Ogrenci *ogr) {
	for (ogrIter = ogrenciler.begin(); ogrIter != ogrenciler.end(); ogrIter++) {
		if ((*ogrIter)->getNo() == ogr->getNo())
			return false;
	}
	ogrenciler.push_back(ogr);
	return true;
}
bool Kayit::yeniDersKaydet(Ders *ders) {
	for (dersIter = dersler.begin(); dersIter != dersler.end(); dersIter++) {
		if ((*dersIter)->getKod() == ders->getKod())
			return false;
	}
	dersler.push_back(ders);
	return true;
}

bool Kayit::ogrenciDerseKaydet(long no, string kod) {
	for (ogrIter = ogrenciler.begin(); ogrIter != ogrenciler.end(); ogrIter++) {
		if ((*ogrIter)->getNo() == no) {
			break;
		}
	}
	for (dersIter = dersler.begin(); dersIter != dersler.end(); dersIter++) {
		if ((*dersIter)->getKod() == kod) {
			break;
		}
	}
	if ((ogrIter != ogrenciler.end()) && (dersIter != dersler.end())) {
		(*ogrIter)->dersEkle((*dersIter));
		(*dersIter)->ogrenciEkle(*ogrIter);
		return true;
	}
	return false;
}

void Kayit::ogrenciListele() {
	for (ogrIter = ogrenciler.begin(); ogrIter != ogrenciler.end(); ogrIter++) {
```

```
			(*ogrIter)->print();
		}

}

void Kayit::DersListele() {
		for (dersIter = dersler.begin(); dersIter != dersler.end(); dersIter++) {
			(*dersIter)->print();
		}

}

void Kayit::dersKayitlari(string kod)
{
		for (dersIter = dersler.begin(); dersIter != dersler.end(); dersIter++) {
			if ((*dersIter)->getKod() == kod) {
				(*dersIter)->listele();
			}
		}

}
void Kayit::kayitliDersler(long no)
{
		for (ogrIter = ogrenciler.begin(); ogrIter != ogrenciler.end(); ogrIter++) {
			if ((*ogrIter)->getNo() == no) {
				(*ogrIter)->listele();
			}
		}
}
```

(c)

KayitApp.cpp

```
int main() {

		Kayit kayit;
		kayit.yeniDersKaydet(new Ders("1001", "Matematik", 4));
		kayit.yeniDersKaydet(new Ders("1002", "Fizik", 3));
		kayit.yeniOgrenciKaydet(new Ogrenci(500, "Ali", "Kemal"));
		kayit.yeniOgrenciKaydet(new Ogrenci(600, "Ayse", "Oz"));
		kayit.yeniOgrenciKaydet(new Ogrenci(700, "Cemal", "Akar"));

		kayit.ogrenciDerseKaydet(600, "1001");
		kayit.ogrenciDerseKaydet(700, "1001");
		kayit.ogrenciDerseKaydet(500, "1002");

		cout << "-----------------Ogrenci Liste---------------" << endl;
		kayit.ogrenciListele();
		cout << "-----------------Ders Liste---------------" << endl;
		kayit.DersListele();
		cout << "-----------------Matemetik Liste---------------" << endl;
		kayit.dersKayitlari("1001");
		cout << "-----------------Fizik Liste---------------" << endl;
		kayit.dersKayitlari("1002");
		cout << "-----------------Ali Kemal Alınanlar---------------" << endl;
		kayit.kayitliDersler(500);

		return 0;
}
```