```
#include <string>

{
	class PairOfShoes
	{
		const std::string name;
		double price;
	public:
		PairOfShoes(const std::string& name, double price);
		PairOfShoes(const PairOfShoes& pair);

		const std::string& GetName() const;
		double GetPrice() const;
		void SetPrice(double newPrice);
	};
}
```

```
#include "PairOfShoes.h"

{
	PairOfShoes::PairOfShoes(const std::string& name, double price)
: name(name), price(price) {}
	PairOfShoes::PairOfShoes(const PairOfShoes& pair) :
PairOfShoes(pair.name, pair.price) {}

	void PairOfShoes::SetPrice(double newPrice)
	{
		this->price = newPrice;
	}

	double PairOfShoes::GetPrice() const
	{
		return this->price;
	}

	const std::string& PairOfShoes::GetName() const
	{
		return this->name;
	}
}
```

```
#include "PairOfShoes.h"
#define DEFAULT_INIT_STORAGE_SIZE 4


{
	class ShoeStorage
	{
		PairOfShoes** shoes; // Array of pointers to PairOfShoes.
Some of them are NULL
		int arraySize;
		int capicity;
	public:
		ShoeStorage(int initStorageSize =
DEFAULT_INIT_STORAGE_SIZE);

		ShoeStorage(const ShoeStorage& storage);
		ShoeStorage& operator=(const ShoeStorage& storage);
		~ShoeStorage();

		void AddPairOfShoes(const PairOfShoes& pair);
		int FindShoeByName(const std::string& shoeName) const;
		void RemovePairOfShoes(const std::string& shoeName);
		double GetPrice(const std::string& shoeName) const;

		double AverageShoePrice() const;
	};
}
```

```
#include "ShoeStorage.h"

{
	ShoeStorage::ShoeStorage(int initStorageSize)
	{
		this->shoes = new PairOfShoes*[initStorageSize];
		for (int i = 0; i < initStorageSize; i++)
			this->shoes[i] = nullptr;

		this->capicity = initStorageSize;
		this->arraySize = 0;
	}

	//copy constructor
	ShoeStorage::ShoeStorage(const ShoeStorage& storage)
	{
		this->shoes = nullptr;		//cirtial
initialize for the operator=, if it isnt set to nullptr,
		*this = storage;		//the program
will try to free unallocated grabage memory.
	}

	//assignment operator performs deep copy
	ShoeStorage& ShoeStorage::operator=(const ShoeStorage& storage)
	{
		if (this == &storage)
			return *this;

		for (int i = 0; i < this->arraySize; i++)
			delete this->shoes[i];
		delete[] this->shoes;

		this->shoes = new PairOfShoes*[storage.capicity];
		this->arraySize = storage.arraySize;
		this->capicity = storage.capicity;

		for (int i = 0; i < this->capicity; i++)
			this->shoes[i] = nullptr;
	//we have to put null in all the cells!

		for (int i = 0; i < this->arraySize; i++)	//now, we
copy only the relevant objects
			if (storage.shoes[i] != nullptr)
				this->shoes[i] = new
PairOfShoes(*(storage.shoes[i]));		//we have to save the
physical object and not adress.

		return *this;
	}

	ShoeStorage::~ShoeStorage()
	{
		for (int i = 0; i < this->arraySize; i++)
			delete this->shoes[i];

		delete[] this->shoes;
		this->shoes = nullptr;
	}
```

```
	void ShoeStorage::AddPairOfShoes(const PairOfShoes& pair)
	{
		if (this->arraySize == this->capicity)
		{
			{
				this->capicity *= 2;
				ShoeStorage temp(*this);
				temp.AddPairOfShoes(pair);

				*this = temp;
			}
			//here temp's dtor is called
		}
		else
			this->shoes[this->arraySize++] = new
PairOfShoes(pair);
	}

	//removes one pair of shoes of a given name
	void ShoeStorage::RemovePairOfShoes(const std::string& shoeName)
	{
		int index = FindShoeByName(shoeName);
		if (index != -1)
		{
			delete this->shoes[index];
			this->shoes[index] = nullptr;
		}
	}

	//returns the price of a given shoe name, returns 0 if shoes is
not found
	double ShoeStorage::GetPrice(const std::string& shoeName) const
	{
		int index = FindShoeByName(shoeName);
		return index == -1 ? 0 : this->shoes[index]->GetPrice();
	//cheking if the shoe was found and return accordingly.
	}

	//returns the index of a pair of shoes by name, returns -1 id
shoes not found
	int ShoeStorage::FindShoeByName(const std::string& shoeName)
const
	{
		for (int i = 0; i < arraySize; i++)
	//in order to prevent duplication, this function will be called
several times.
			if (this->shoes[i] != nullptr &&
				this->shoes[i]->GetName() == shoeName)
				return i;

		return -1;
	}


	double ShoeStorage::AverageShoePrice() const
	{
		double price = 0;
```

```
		int numberOfShoes = 0;

		for (int i = 0; i < arraySize; i++)
		{
			if (this->shoes[i] != nullptr)
			{
				numberOfShoes++;
				price += this->shoes[i]->GetPrice();
			}
		}

		return price > 0 ? price / numberOfShoes : 0;
	//if price > 0 then also numberOfShoes
	}
}
```

```
#include "ShoeStorage.h"

#define DEFAULT_DISCOUNT 0.0

{
	class ShoeStore
	{
		ShoeStorage shoeStorage;
		double currentDiscountPrecent;
		double calculateDiscount(double price) const;

	public:
		ShoeStore(double discountPercent = DEFAULT_DISCOUNT);

		void SetDiscountPercent(int discount) {
currentDiscountPrecent = discount; }
		double GetDiscountPercent() const { return
currentDiscountPrecent; }

		void AddShoes(const std::string& name, double price, int
amount = 1);
		void RemoveOnePair(const std::string& name);

		double AverageShoePrice() const;
		double GetShoePrice(const std::string& shoeName) const;
	};
}
```

```
#include "ShoeStore.h"

{

	ShoeStore::ShoeStore(double discountPercent) :
		shoeStorage(), currentDiscountPrecent(discountPercent) {
}

	//adds shoes to the storage
	void ShoeStore::AddShoes(const std::string& name, double price,
int amount)
	{
		for (int i = 0; i < amount; i++)
			this-
>shoeStorage.AddPairOfShoes(PairOfShoes(name, price));		//if
we will send reference, after the function ends it will be nullptr
	}

	//remove one pair from the storage
	void ShoeStore::RemoveOnePair(const std::string& name)
	{
		this->shoeStorage.RemovePairOfShoes(name);
	}

	//returns the average shoes price after discount
	double ShoeStore::AverageShoePrice() const
	{
		double price = this->shoeStorage.AverageShoePrice();

		return calculateDiscount(price);
	}

	//returns the shoe price after discount
	double ShoeStore::GetShoePrice(const std::string& shoeName)
const
	{
		double price = this->shoeStorage.GetPrice(shoeName);
		return calculateDiscount(price);
	}

	//given a price, this method calculates the price after discount
	double ShoeStore::calculateDiscount(double price) const
	{
		return price - price * currentDiscountPrecent / 100;
	}
}
```