T.C.
İSTANBUL ÜNİVERSİTESİ
Mühendislik Fakültesi
Bilgisayar Mühendisliği Bölümü

Dersin Kodu: BIMU1055	**Dersin Adı: INTRODUCTION TO PROGRAMMING**
Dersin Öğretim Üyesi: Dr. Öğr. Ü. Özgür Can TURNA	**Sınav Türü: Final**
Sınav Tarihi ve Süresi: 22.05.2018 (70 dk)	**Öğrenci No:**
Öğrenci Ad-Soyad:	**Öğrenci İmzası:**

This class definition is used in questions 1-4.

```
#include <iostream>
using namespace std;
class Parent
{
protected:
      int pint;
};

class Child : public Parent
{
public:
      int getPval() { return pint; }
      void setPval(int v) { pint = v; }
private:
      int cint;
};
```

1. (10p) Add necessary code lines to Parent and Child class to be able to write this Stranger class.

```
class Stranger
{
public:
      void setP(Parent &p, int v) { p.pint = v; }
      void setC(Child &c, int v) { c.cint = v; }
};
```

Parent: `friend Stranger;`

Child : `friend Stranger;`

2. (10p) How can you disable other programmers to generate a copy of Parent class but allow to generate copies of Child class in other class implementations?
   Ex: in main:
   `Parent p1; //Forbidden`
   `Child c1; //Allowed`
   Write necessary code lines for Parent & Child classes in brackets.

```
Parent:     protected:       //By hiding Default constructor
                Parent(){}   //or  you cane crate a pure virtual function in Parent
Child:      public:
                Child(){}    //Make child constructor available for everyone
```

3. (15p) Write necessary code lines for Child class to give this functionality?

```
int main()
{
	Child c1(5), c2(10);
	c1 = c1 + c2;
	cout << "c1.pint:" << c1.getPval();   // returns 15
}
```

```
In Child class:
public:
	Child(int a) { setPval(a); }
	Child operator+(Child &c2) { return Child(getPval() + c2.getPval()); }
```

4. (15p) Write necessary code lines for Parent and Child class to give this functionality? Change any function if necessary.

```
class ChildType2 : public Parent{ };
int main()
{
	Child c1, c2;
	ChildType2 ct2;
	cout << Parent::count;    // Gives 3 as output
}
```

```
class Parent
{
protected:
	int pint;
	Parent() { count++; }
public:
	static int count;
};
int Parent::count = 0;
```

5. (20p) Write a function to find any given char in a char array. If the array does not include the given char your function must throw a custom exception that you implemented and you must catch this exception where you call this function in main.

```
#include <iostream>
#include <string>
using namespace std;
bool finder(char * str , char key)
{
	for (int i = 0; str[i] != '\0' ; i++)
	{
		if (str[i] == key)
			return true;
	}
	throw exception(("There is no copies of"+ key + (string)" in " + str).c_str() );
}
int main()
{
	char arr[] = "Hello is there any of ! in me :P";
	char key = '?';
	bool not_enough = false;
	try {
		not_enough = finder(arr, key);
	}
	catch (std::exception)
	{
		cerr << "failed to find a copy in string" << endl;
	}
	cout << "result: " << not_enough;
	return 0;
}
```

6. (30p) Create a class that can store any type of objects (int , float or your data types) and has the functionalities below.

```
int main()
{
  Storage<int> myStorage;
  myStorage.add(10);    myStorage.add(20);      myStorage.add(10);
  cout << myStorage.size() << endl; // Output : 3
  if (myStorage.remove(30))
      cout << "We cleaned all 30 in storage" << endl; //NOT prineted
  if (myStorage.remove(10))
      cout << "We cleaned all 10 in storage" << endl; //PRINTED
  cout << myStorage.size() << endl; // Output : 1
}
```

```cpp
#include <iostream>
#include <string>
using namespace std;
template<typename T>
class listNode
{
public:
	T value;
	listNode * next;
	listNode(T v) : value(v) { next = NULL; }
};
template<typename T>
class Storage {
private:
	listNode<T> * head;
	int listSize = 0;
public:
	Storage() { head = NULL; }
	void add(T node) {
		listNode<T> *newNode = new listNode<T>(node);
		newNode->next = head;
		head = newNode;
		listSize++;
	}
	int size() {
		return listSize;
	}
	bool remove(T node) {
		listNode<T> *ptr = head, *prev = NULL;
		bool occurs = false;
		while (ptr != NULL )
		{
			prev = ptr;
			ptr = ptr->next;
			if (prev->value == node)
			{
				head = ptr;
				listSize--;
				occurs = true;
			}
			else if(ptr != NULL && ptr->value == node)
			{
				prev->next = ptr->next;
				ptr = ptr->next;
				listSize--;
				occurs = true;
			}
		}
		return occurs;
	}
};
```