MS TEAMS - Session Notes (06-APR-2020)

Examples: The following code includes the different types of objects we can create.

```
//Automatic Objects: Objects created within the local scope of a function { }. They are destructed
//at the end of the local scope automatically.
//Static Objects: Objects created inside the local scope and having local visibility, but
//persisting from their declaration till the end of the program.
//External Objects: Objects created outside any function.

//test.h
class test{
private:
	int num;
public:
	test()
	{
		num = 0;
	}
	void add_number(int num_in)
	{
		this->num += num_in;
	}
	int getnum()
	{
		return this->num;
	}
	~test()
	{
		cout << "The object with number:" << this->num << " has been deleted."<<endl;
	}
};
//external object
test obj1;

//test.cpp
#include<iostream>
using namespace std;
#include"test.h"
void main()
{
	obj1.add_number(5);

	for (int i = 0; i < 3; i++)
	{
		test obj2; //automatic object
		obj2.add_number(10);
```

```
		static test obj3; //static object
		obj3.add_number(15);

		cout << "Automatic object has the number:" << obj2.getnum() << endl;
		cout << "Static object has the number:" << obj3.getnum() << endl;
	}//1 obj2 (automatic)
	cout << "External object has the number:" << obj1.getnum() << endl;

	system("pause");
}//2 obj3 (static)
//external object (obj1) will be destructed once the whole project/VS is closed/terminated.
```

Example 2: Constant members and objects

```
//constex.h
class constex{
private:
	int x;
public:
	constex(int x)
	{
		this->x = x;
	}
	void setX(int x)
	{
		this->x = x;
	}
	int getX() const
	{
		return this->x;
	}
};

//constex.cpp
#include<iostream>
using namespace std;
#include"constExample.h"
void main()
{
	constex obj1(5);
	const constex obj2(10);
	obj2.setX(20); // are we allowed to modify const objects??? NO
	cout << obj1.getX()<<endl; //are we allowed to call const member functions using non-const
objects? YES
	system("pause");
} //see Lecture2 for more details
```

Example 3: Pointers

```
//ptrex1.cpp
#include<iostream>
using namespace std;
void main()
{
	int x = 25;
	int *ptr; // we must initialize this pointer variable!!!
	//assign (=) the address (&) of variable x to ptr.
	ptr = &x;

	cout << "The value of x:" << x << " or " << *ptr << endl;
	cout << "The address of x is:" << ptr <<" or "<<&x<< endl;
	system("pause");

}

//ptrex2
#include<iostream>
using namespace std;
void main()
{
	int x = 7, *xptr;
	xptr = &x;
	cout << "The address of x is:" << &x << endl;
	cout << "The value of xptr is:" << xptr << endl;
	cout << "The value of x is:" << x << endl;
	cout << "The value of *xptr is:" << *xptr << endl;
	cout << &*xptr << " " << *&xptr << endl; //* and & operators are inverses of each other.
Means
                                        //when we use them togehter they cancel each other.

	system("pause");
}
```

Example 4: How to work with pointers: Creating objects using pointers

```
//rectangle.h
class rectangle{
private:
	int length, width;
public:
	rectangle()
	{
		length = 1;
		width = 1;
	}
```

```
	rectangle(int length, int width)
	{
		this->length = length;
		this->width = width;
	}
	void setdimensions(int length, int width)
	{
		this->length = length;
		this->width = width;
	}
	int getLength() const
	{
		return this->length;
	}
	int getWidth() const
	{
		return this->width;
	}
};

//rectangle.cpp
#include<iostream>
using namespace std;
#include"rectangle.h"
void main()
{
	//create an object using a pointer
	rectangle robj1, *ptrobj1;
	ptrobj1 = &robj1; //initializing the ptrobj1 with the address of robj1 object
	//length=1, width=1
	ptrobj1->setdimensions(10, 20);
	//or
	(*ptrobj1).setdimensions(20, 30);
	cout << ptrobj1->getLength() << " " << ptrobj1->getWidth() << endl;

	rectangle arrobj[5], *arrptr;
	arrptr = arrobj;
	for (int i = 0; i < 5; i++)
	{
		arrptr[i].setdimensions(1, 2);
		cout << arrptr[i].getLength() << " " << arrptr[i].getWidth() << endl;
	}

	arrptr[0].setdimensions(20, 30);

	cout << endl;
```

```
	for (int i = 0; i < 5; i++)
		cout << arrptr[i].getLength() << " " << arrptr[i].getWidth() << endl;

	//array object that invokes the parameterized constructor
	rectangle arr_rect[5] = { rectangle(1, 2), rectangle(2, 3), rectangle(3, 4), rectangle(5, 6),
rectangle(7, 8) };

	system("pause");
}
```