Course: CSC 340.03(TIC)
Student: Tin Thu Zar Aye
Student ID: 920615641
Assignment Number: 03
Assignment Due Date : 04-25-2020

1. Deleting the same memory twice is the undefined behavior. If we delete the same memory twice, the first delete will not give any errors but the second delete will definitely give the error. It can also crush the program. It also can change the object which already out the heap.

```
#include<iostream>
using namespace std;
int main()
{
	cout<<"Deleting the same memory twice" << endl;

	int *ptr = = new int;
	delete ptr;
	delete prt; // can cause the error

}
```

2. The smart pointer is only taking care of the deleting the memory when there is no remaining smart pointers pointing to that memory. Smart pointer can provide safety and convenience for handling dynamically allocated memory only. The smart pointer often saves you the need to do things explicitly. In the data member of classes, when an object is deleted all the owned data is deleted as well without any special code in the destructor. The smart pointer contains the call to delete because the smart pointer is declared in stack.

```
#include<iostream>
using namespace std;
int main()
{
	unique_ptr<person>personPtr;

	personPtr name1;

	personPtr name2(new person());

	name1 = name2;
}
```

```
//after copying the pointer, there are 2 references to the
object. Then leaving one object after name2 is destroyed. After
that name1 is destroyed and leave a reference count to 0.
```

3. The smart pointer uses to point to the object of class. It has own member function which can access by using dot notation. The smart pointer deletes the object. When the object is going out of scope, the memory for allocated data will delete automatically.

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

class Name
{
private:
     string firstName {"N/A"};
     string lastName { "N/A"};

public:
     Name();
     Name(const string &, const string &);
     ~Name();

     string getName() const;
};

Name::Name{}

Name::Name(const string& firstName, const string& lastName):
firstName(firstName), lastName(lastName){}

Name::~Name()
{
     cout<<"Destructor" << this << "," << this->getName() <<
endl;
}

string Name::getName()
{
     return this->firstName + " " + this->secondName;
}

int main()
{
```

```
    unique_ptr<Name>objPtr; // declare smart pointer pointing
to the object of a class
}
//the memory for allocated data will delete when the object goes
out of scope.
```

4. Both unique pointer and shared pointer are smart pointers. The unique pointer can be converted to the shared pointer because unique pointer owned an object exclusively. After that, the unique pointer doesn't own the object anymore. Moreover, the unique pointer is more efficient than a shared pointer, so when you start with the unique pointer, you can always convert to the shared pointer anytime.

```
#include<iostream>
using namespace std;
int main()
{
    unique_ptr<Name>goofy_unique{make_unique<Name>("Goofy",
"Dog")};

    // converting from unique pointer to shared pointer by
using move()

    shared_ptr<Name>goofy_shared{move(goofy_unique)};
}
```

5. A weak pointer isn't a standalone smart pointer. It's an augmentation of shared pointer. A weak pointer is created from shared pointer and they both are pointing to the same place as the shared pointer initializing them but they don't affect the reference count of the object they point to. A pointer like shared pointer doesn't affect an object's. reference count. If the weak pointer has expired and if not gives you access to the object to, then this is done by creating shared pointer from the weak pointer. For example, person A and person B are containing shared pointer and they are pointing to each other. If person A destroyed, person B's pointer back to dangle but person B's pointer won't affect A's reference count.

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

```
int main()
{
     Name* mickey = new Name("Mickey", "Mouse");
     Name *mickeyPtr{mickey};
     delete mickey;
     mickey = nullptr;

     shared_ptr<Name>goofy(new Name{"Goofy", "Dog"});
     weak_ptr<Name>goofy_wptr{goofy};

     cout<<(goofy_wptr.expired() ?) << endl; // goofy_wptr is
not dangling yet.

     goofy.reset();

     cout<<(goofy.wptr.expired() ?) << endl; // now goofy_wptr
is dangling.

}
```