Week 4-1 : Constructor, Destructor

Part1. Constructor

- Form of the Constructor

Get a same name with class name
there is no return type, truly return nothing.
One of the function, possibly set the overloading and default value.

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

class Constructor
{
int num1;
int num2;

public:
Constructor()
{
num1=0;
num2=0;
}
Constructor(int n)
{
num1=n;
num2=0;
}
Constructor(int n1, int n2)
{
num1=n1;
num2=n2;
}

/* default parameter constructor
Constructor(int  n1=0,  int n2=0)
{
num1=n1;
num2=n2;
}
*/

void  ShowData() const
{
cout<<num1<<' '<<num2<<endl;
}
};

int main(void)
{
Constructor sc1;
sc1.ShowData();

Constructor sc2(100);
sc2.ShowData();

Constructor sc3(100, 200);
sc3.ShowData();
return 0;
}
```

When sc1, sc2, sc3 objects are being made, they pass overloaded constructor. If you use the defualt parameter constructor, then you erase other contructors. the result is same.

- The initialization using member initializer

Use member initializer when you call constructors of the objects which is declared as member variable.
Not initialize at the body, initialize at the next of the parameters.

```
#include <iostream>
```

```
using  namespace std;

class Constructor
{
int num1;
int num2;

public:

Constructor(int n1, int n2) : num1(n1),     num2(n2)
{

}

void  ShowData() const
{
cout<<num1<<' '<<num2<<endl;
}
};

int main(void)
{
Constructor sc(100,200);
sc.ShowData();

return 0;
}
```

Part2. Destructor

Destruct the resources which is allocated by constructor.
If there is memory space allocated by new operator, then destructor destruct this memory space.

reference>> new and delete

They are compared to malloc and free respectively.
When you generate objects, you have to use "new".

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

class Book
{
private:
char * bookName;
int bookNum;
public:
Book(char  *  tempName,  int tempNum)
{
int len=strlen(tempName)+1;
bookName=new char[len];
strcpy(bookName, tempName);
bookNum=tempNum;
}

void  ShowBookInfo() const
{
cout<<"Book Name : "<<bookName<<endl;
cout<<"Book  Number  : "<<bookNum<<endl;
}

~Book()
{
delete []bookName;
cout<<"destructor"<<endl;
}
};

int main(void)
{
Book book1("Computer Programming", 2001001);
Book book2("This is C++", 400010);
book1.ShowBookInfo();
book2.ShowBookInfo();
return 0;
}
```

Week 4-2 : Copy Constructor, Vector, Polymorphism

Part3. Copy Constructor

- Copy Constructor

When you recall name which is generated in parameter, copy the object.

If you are not definite copy constructor, default copy constructor insert automatically.

- Kinds of copy constructor through conversions

implicit conversion : = , explicit conversion : (object)

you have to use explict to prevent implicit conversion

- Time of calling copy constructor

1. Initialize a new object using a already generated object.

```
Point x2(x1);
```

2. Call-by-value : pass the object as a parameter during the function calling

```
Point copyFunc(Point obj)
{

        return obj;
}
```

3. return the object which is not returned by the references.

```
Point copyFunc(Point obj)
{
        return obj;
}
```

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

class Book
{
private:
char * bookName;
int bookNum;
public:
Book(char  *  tempName,  int tempNum)
{
int len=strlen(tempName)+1;
bookName=new char[len];
strcpy(bookName, tempName);
bookNum=tempNum;
}

void  ShowBookInfo() const
{
cout<<"Book Name : "<<bookName<<endl;
cout<<"Book  Number  : "<<bookNum<<endl;
}

~Book()
{
delete []bookName;
cout<<"destructor"<<endl;
}
};

int main(void)
{
Book book1("Computer Programming", 2001001);
Book  book2("This  is  C++", 400010);
Book book3(book2);
book1.ShowBookInfo();
book2.ShowBookInfo();
book3.ShowBookInfo();
return 0;
}
```

if you not define any copy constructor, a default copy constructor copys member to member. Upper code has an error, the default copy constructor points same book name part, destructor destruct at book2, and destructor destruct at book3, too. But there is nothing to destruct. because string already destructed. so, error appears.

To solve this problem, it needs to copy this book name part into another memory. this is called "deep copy".

```
Book(const Book& copy) : bookNum(copy.bookNum)
{
bookName = new char[strlen(copy.bookName)+1];
strcpy(bookName, copy.bookName);
}
```

- **Vector**

An array-based container that supports a random access iterator.
Elements are stored consecutively in one memory block.

```
template<typename T, typename Allocator = allocator<T>>
class vector
```

v.pop_back() : Remove the last element of v.
v.push_back() : Add the element to the end of v.

```
#include <iostream>
#include <vector>

using namespace std;

int main(void)
{
vector<int> v;
v.push_back(5);
v.push_back(2);
v.pop_back();
}
```

- Polymorphism

Same sentence but different result.
Polymorphism : the method of implementing all of the super -class' member. Sub-class has its own member and super class' member.
Is-a relation.

- Polymorphism

```
class Person
{
private:
            int age;
            char name[50];
public:
            Person(int myage, char * myname) :    age(myage)
            {
            strcpy(name, myname);
            }
            void  ShowName() const
            {
            cout<<"My  name is"<<name<<endl;
            }
            void  ShowAge() const
            {
            cout<<"My  age is"<<age<<endl;
            }
};

class  Student  :  public Person
```

```
{
private:
		char major[50];
public:
		Student(char * myname, int myage, char * mymajor) : Person(myage, 
myname)
		{
		strcpy(major, mymajor);
		}
		void  ShowStudent() const
		{
		ShowName();
		ShowAge();
		cout<<"My  major is"<<major<<endl<<endl;
		}
};
```

Student is a person. (is-a relation), student class inherits person class. Student is implemented by 'public Person'. And Student is inherited from Person's member.