this pointer, static & const and Operator Overloading (12th Lab)

✧ 'this' pointer

- Self-reference pointer
- A pointer used to reference itself
- Used in member function
- A pointer that has non-uniform value
  - If 'this' is used in the object allocated to 0x10, it would be a pointer that means 0x10.
  - If 'this' is used in other member function, it would be another value.

```cpp
#include <iostream>
using std::cout;
using std::endl;

class ThisPtr{
 public:
   ThisPtr* GetThis() {
      return this; // return this pointer
   }
};

int main() {
   ThisPtr *p1 = new ThisPtr(); // create ThisPtr Object
   cout << "Pointer p1" << p1 << endl;
   cout << "This pointer of p1" << p1->GetThis() << endl;
   getchar();
   return 0;
}
```

- result

```
C:\Windows\System32\cmd.exe
Pointer p100387f90
This pointer of p100387f90
```

DCSLab
CSE, SNU
```cpp
#include <iostream>
using std::cout;
using std::endl;

class Data {
    int a;
    int b;

public:
    Data(int a, int b) {
        // a = a;
        this->a = a;

        // b = b;
        this->b = b;
    }
    void printAll() {
        cout << a << " " << b << endl;
    }
};

int main() {
    Data d(100, 200);
    d.printAll();
    return 0;
}
```

**result**

- a = a;

  - Left value ‘a’ is not a member value, but parameter value ‘a’
  - Member variable cannot be preferred over the local variable(and also parameter)
- In constructor, it cannot access the member variable by using the name ‘a’
- this->a = a;
- using this pointer
- possible to access member variable

**const**
- Identifier making constant
- Making data pointed by pointer constant
- Making pointer itself constant
  - Cannot change the pointing value
- Making member function constant
  - Cannot allow the member variable changed
- Making member variable of class constant
  - Need member initializer

```cpp
class Student{
    const int id;
    char name[20];
    char major[30];
public:
    Student(int _id, char* _name, char* _major) : id(_id)
    {
        // id = _id;
        strcpy(name, _name);
        strcpy(major, _major);
    }
};
```
static member is initialized on the memory space, before the main function is called.
- Like a global variable
- static member initialization execution

```cpp
class AAA{
public:
    static int n;
};

int AAA::n = 1;

int main(){
    cout<<AAA::n<<endl;
    AAA::n++;
    cout<<AAA::n<<endl;

    return 0;
}
```

- static member initialization : int AAA::n = 1;
- after initialization, then referencing is possible

```cpp
#include <iostream>
using std::cout;
using std::endl;

class Person{
    char name[20];
    int count;

public:
    Person(char* _name){
        count = 1;
        strcpy(name, _name);
        cout<<count++<<": Person created"<<endl;
    }
```
int main(){
    Person p1("Minyoung");
    Person p2("Dongyou");
    getchar();
    return 0;
}

result

- All Person object should share the variable count, in above example there exists the independent count variable in each created object.
- Count value is not increased.

class Person{
    char name[20];
    static int count;
public:
    Person(char* _name){
        //count = 1;
        strcpy(name, _name);
        cout<<count++<<": Person created"<<endl;
    }
}

int Person::count=1;
int main(){
    Person p1("Minyoung");
    Person p2("Dongyou");
    getchar();
    return 0;
}
Operator Overloading

Operators that can be overloaded

+  -  *  /  %  ^  &  |  -=  /=  <<=  ==  !=  <=  >=  &&  ||  ++  --  ->  *  ,  [ ]  ()  new  delete

Operators that cannot be overloaded

.  .*  ::  ?:

- increment(++)/decrement(--) operators

class Point{
private:
    int x,y;
public:
    Point(int _x=0, int _y=0):x(_x), y(_y)
    {}
    void ShowPosition();
    Point& operator++();
    friend Point& operator--(Point& p);
};
```cpp
void Point::ShowPosition() {
    cout << x << " " << y << endl;
}
Point& Point::operator++() { // member function overloading
    x++;
    y++;
    return *this;
}
Point& operator--(Point& p) { // Global variable overloading
    p.x--;
    p.y--;
    return p;
}
int main()
{
    Point p(1,2);
    ++p;
    p.ShowPosition(); // 2 3
    --p;
    p.ShowPosition(); // 1 2
    ++(++p);
    p.ShowPosition(); // 3 4
    --(--p);
    p.ShowPosition(); // 1 2
    return 0;
}
```

1. `return *this;`
   - this : pointer is pointing itself
   - *this : reference the pointing value = return itself
2. `Point& operator++();`
   - Return type : Point&
   - If Point is return type, ++(++p) value increase by 1. Because
     returned value by ++p operation is not reference of p, but copy of p.

- To distinguish prefix and postfix increment
  - ++p : p.operator++();
- \texttt{p++ : p.operator++(int)};

\begin{itemize}
  \item return values
    \begin{itemize}
      \item prefix increment
        \begin{itemize}
          \item returns by reference
        \end{itemize}
      \item postfix increment
        \begin{itemize}
          \item returns by value
          \item returns temporary object
        \end{itemize}
    \end{itemize}
\end{itemize}

```cpp
class Point{
private:
    int x, y;
public:
    Point(int _x = 0, int _y = 0): x(_x), y(_y) {};
    void ShowPosition();
    Point& operator++();
    Point operator++(int);
};
void Point::ShowPosition(){
    cout << x << " " << y << endl;
}
Point& Point::operator++(){
    x++;
    y++;
    return *this;
}
Point Point::operator++(int){
    Point temp(x, y);
    x++;
    y++;
    return temp;
}
int main(){
    Point p1(1, 2);
    (p1++).ShowPosition();
    p1.ShowPosition();
    Point p2(1, 2);
```
cout, cin overloading
- cout : ostream object/ cin : istream object
- already overloaded to process each built-in type
- can also process a user-defined class
  ◆ overload using global, friend functions

```cpp
#include <iostream>
using std::endl;
using std::cout;
using std::ostream;

class Point{
private:
    int x,y;
public:
    Point(int _x=0, int _y=0):x(_x), y(_y) {}  
    friend ostream& operator<<(ostream& os, const Point& p);  
};

ostream& operator<<(ostream& os, const Point& p){
    os<<"("<<p.x<<","<<p.y<<")"<<endl;
    return os;
}

int main() {
    Point p(1,3);  
    cout<<p;  
    return 0;
}
```
Practice

- Print out formatted number automatically using operator overloading
  - format: (123) 456-7890
  - you should use setw() to align the number
    - input >> setw(3) >> number.areaCode;
  - (,), space is ignored
    - input.ignore(2); // skip ) and space

```cpp
// PhoneNumber.h

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

class PhoneNumber
{
    friend ostream &operator<<( ostream &, const PhoneNumber & );
    friend istream &operator>>( istream &, PhoneNumber & );

private:
    string areaCode; // 3-digit area code
    string exchange; // 3-digit exchange
    string line; // 4-digit line
}; // end class PhoneNumber

// PhoneNumber.cpp

#include <iomanip>
#include "PhoneNumber.h"
using namespace std;

// overloaded stream insertion operator; cannot be
// a member function if we would like to invoke it with
```
// cout << somePhoneNumber;
ostream &operator<<( ostream &output, const PhoneNumber &number )
{
    output << "(" << number.areaCode << "\"\"
    << number.exchange << "-\"" << number.line;
    return output; // enables cout << a << b << c;
} // end function operator<<

// overloaded stream extraction operator; cannot be
// a member function if we would like to invoke it with
// cin >> somePhoneNumber;
istream &operator>>( istream &input, PhoneNumber &number )
{
    input.ignore(); // skip ( input.ignore( 3 ) >> number.areaCode: // input area code
    input.ignore( 2 ); // skip ) and space
    input >> setw( 3 ) >> number.exchange: // input exchange
    input.ignore(); // skip dash (-)
    input >> setw( 4 ) >> number.line: // input line
    return input; // enables cin >> a >> b >> c;
} // end function operator>>

#include <iostream>
#include "PhoneNumber.h"
using namespace std;

int main()
{
    PhoneNumber phone; // create object phone

    cout << "Enter phone number in the form (123) 456-7890:" << endl;

    // cin >> phone invokes operator>> by implicitly issuing
    // the global function call operator>>( cin, phone )
```cpp
    cin >> phone;
    
    cout << "The phone number entered was: ";
    // cout << phone invokes operator<< by implicitly issuing
    // the global function call operator<<( cout, phone )
    cout << phone << endl;
    
    return 0;
} // end main
```