const 함수를 이용한 overloading의 활용
(StableConstArrayProb.cpp 1/3)

459-463p

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

class BoundCheckIntArray
{
private:
        int *arr;
        int arrlen;

        BoundCheckIntArray(const BoundCheckIntArray& arr) { }
        BoundCheckIntArray& operator=(const BoundCheckIntArray& arr) { }

public:
        BoundCheckIntArray(int len);
        int& operator[] (int idx);
        int& operator[] (int idx) const;
        int GetArrLen() const;
        ~BoundCheckIntArray();
};
```

[실행결과]
11
22
33
44
55

복사 생성이나 대입 연산을 막음.

const에 의한 연산자 overloading:
이 함수가 없으면 이 클래스의 const 객체에 대해서는 [] 연산 사용 불가함.
(가려 놓고 테스트 해보기!)

459-463p

const 함수를 이용한 overloading의 활용
(StableConstArrayProb.cpp 2/3)

```
void ShowAllData(const BoundCheckIntArray& ref)
{
        int len = ref.GetArrLen();

        for (int idx = 0; idx < len; idx++)
                cout << ref[idx] << endl;
}

int main(void)
{
        BoundCheckIntArray arr(5);

        for (int i = 0; i < 5; i++)
                arr[i] = (i + 1) * 11;

        ShowAllData(arr);
        return 0;
}
```

const [ ] 연산자 overloading 이 없으면 여기서 [ ] 연산 사용 불가함.

459-463p

const 함수를 이용한 overloading의 활용 (StableConstArrayProb.cpp 3/3)

```
BoundCheckIntArray::BoundCheckIntArray(int len) :arrlen(len) {
        arr = new int[len];
}
int& BoundCheckIntArray::operator[] (int idx) {
        if (idx < 0 || idx >= arrlen) {
                cout << "Array index out of bound exception" << endl;
                exit(1);
        }
        return arr[idx];
}
int& BoundCheckIntArray::operator[] (int idx) const {
        if (idx < 0 || idx >= arrlen) {
                cout << "Array index out of bound exception" << endl;
                exit(1);
        }
        return arr[idx];
}
int BoundCheckIntArray::GetArrLen() const { return arrlen; }
BoundCheckIntArray::~BoundCheckIntArray() { delete [] arr; }
```

const에 의한 연산자 overloading:
이 함수가 없으면 이 클래스의 const 객체에 대해서는 [] 연산 사용 불가함.
(가려 놓고 테스트!)

객체의 저장을 위한 배열 클래스
(StablePointObjArray.cpp 1/3)

463-465p

◆ int 형 대신 클래스 객체를 저장하는 배열클래스

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

class Point
{
private:
        int xpos, ypos;
public:
        Point(int x = 0, int y = 0) : xpos(x), ypos(y) {  }
        friend ostream& operator<<(ostream& os, const Point& pos);
};

ostream& operator<<(ostream& os, const Point& pos)
{
        os << '[' << pos.xpos << ", " << pos.ypos << ']' << endl;
        return os;
}
```

객체가 배열로 선언될 때는 인자 없는 생성자를 사용하게 되므로 default constructor를 반드시 준비!

객체의 저장을 위한 배열 클래스 (StablePointObjArray.cpp 2/3)

463-465p

```
class BoundCheckPointArray {
private:
    Point* arr;
    int arrlen;
    BoundCheckPointArray(const BoundCheckPointArray& arr) { }
    BoundCheckPointArray& operator=(const BoundCheckPointArray& arr) { }
public:
    BoundCheckPointArray(int len) :arrlen(len) { arr = new Point[len]; }
    Point& operator[] (int idx) {
        if (idx < 0 || idx >= arrlen) {
            cout << "Array index out of bound exception" << endl;
            exit(1);
        }
        return arr[idx];
    }
    Point operator[] (int idx) const {
        if (idx < 0 || idx >= arrlen) {
            cout << "Array index out of bound exception" << endl;
            exit(1);
        }
        return arr[idx];
    }
    int GetArrLen() const { return arrlen; }
    ~BoundCheckPointArray() { delete[]arr; }
};
```

객체가 배열로 선언될 때는 인자 없는 생성자를 사용하게 된다,

463-465p

객체의 저장을 위한 배열 클래스
(StablePointObjArray.cpp 3/3)

```
int main(void)
{
        BoundCheckPointArray arr(3);
        arr[0] = Point(3, 4);
        arr[1] = Point(5, 6);
        arr[2] = Point(7, 8);

        for (int i = 0; i < arr.GetArrLen(); i++)
                cout << arr[i];

        return 0;
}
```

Point에 대입연산자가 없으므로 default 대입연산자가 멤버들을 얕은 복사.

객체의 포인터 저장을 위한 배열 클래스
(StablePointPtrArray.cpp 1/3)

465-467p

- 깊은복사/얕은 복사에 신경 쓰지 않아도 됨.
- 빠른 추가/삭제가 가능.

하므로 객체배열보다 더 많이 사용됨.

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

class Point {
private:
        int xpos, ypos;
public:
        Point(int x = 0, int y = 0) : xpos(x), ypos(y) {  }
        friend ostream& operator<<(ostream& os, const Point& pos);
};

ostream& operator<<(ostream& os, const Point& pos) {
        os << '[' << pos.xpos << ", " << pos.ypos << ']' << endl;
        return os;
}

typedef Point* POINT_PTR;
```

저장의 주 대상이 포인터인 경우 타입 선언 해주는 것이 좋다.

객체의 포인터 저장을 위한 배열 클래스
(StablePointPtrArray.cpp 2/3)

465-467p

```
class BoundCheckPointPtrArray {
private:
    POINT_PTR* arr;
    int arrlen;
    BoundCheckPointPtrArray(const BoundCheckPointPtrArray& arr) { }
    BoundCheckPointPtrArray& operator=(const BoundCheckPointPtrArray& arr) { }
public:
    BoundCheckPointPtrArray(int len) :arrlen(len) { arr = new POINT_PTR[len]; }
    POINT_PTR& operator[] (int idx) {
        if (idx < 0 || idx >= arrlen) {
            cout << "Array index out of bound exception" << endl;
            exit(1);
        }
        return arr[idx];
    }
    POINT_PTR operator[] (int idx) const {
        if (idx < 0 || idx >= arrlen) {
            cout << "Array index out of bound exception" << endl;
            exit(1);
        }
        return arr[idx];
    }
    int GetArrLen() const { return arrlen; }
    ~BoundCheckPointPtrArray() { delete[]arr; }
};
```

객체의 포인터 저장을 위한 배열 클래스
(StablePointPtrArray.cpp 3/3)

465-467p

```
int main(void)
{
        BoundCheckPointPtrArray arr(3);
        arr[0] = new Point(3, 4);
        arr[1] = new Point(5, 6);
        arr[2] = new Point(7, 8);

        for (int i = 0; i < arr.GetArrLen(); i++)
                cout << *(arr[i]);

        delete arr[0];
        delete arr[1];
        delete arr[2];
        return 0;
}
```

주소를 저장하므로 객체 복사는 이루어지지 않음.

494-498p

OOP 프로젝트 08단계

강의자료실에서 "OOP_step_7.zip" 다운로드 받아 수정.

- **Account 클래스**
  - 대입연산자 추가 : 깊은 복사 되도록 정의
- **AccountHandler 클래스**
  - 배열 멤버를 BoundCheckPointPtrArray 배열 클래스로 대체
- **AccountArray.h, AccountArray.cpp**
  - 배열클래스 추가를 위해 추가되는 소스

494-498p

OOP 프로젝트 08단계
(Account.h, Account.cpp 수정)

[Account,h – 대입연산자 추가]

```
Account& operator=(const Account& ref);
```

[Account,cpp – 대입연산자 추가]

```
Account& Account::operator=(const Account& ref)
{
        accID=ref.accID;
        balance=ref.balance;

        delete []cusName;
        cusName=new char[strlen(ref.cusName)+1];
        strcpy(cusName, ref.cusName);
        return *this;
}
```

494-498p

OOP 프로젝트 08단계
(AccountHandler.h 수정)

[배열 제외시키고 배열 클래스를 멤버로 추가]

```
#ifndef __ACCOUN_HANDLER_H__
#define __ACCOUN_HANDLER_H__

#include "Account.h"
#include "AccountArray.h"

class AccountHandler {
private:
        Account * accArr[100];
        BoundCheckAccountPtrArray accArr;
        int accNum;
public:
        AccountHandler();
        void ShowMenu(void) const;
        void MakeAccount(void);
        void DepositMoney(void);
        void WithdrawMoney(void);
        void ShowAllAccInfo(void) const;
        ~AccountHandler();
protected:
        void MakeNormalAccount(void);
        void MakeCreditAccount(void);
};
```

494-498p

OOP 프로젝트 08단계
(BankingCommonDecl.h 수정 <- 수정할 필요없음)

[헤더파일 include문 추가]

```
#ifndef __BANKING_COMMON_H__
#define __BANKING_COMMON_H__

#include <iostream>
#include <cstring>
#include <cstdlib>

using namespace std;
const int NAME_LEN=20;

// 프로그램 사용자의 선택 메뉴
enum {MAKE=1, DEPOSIT, WITHDRAW, INQUIRE, EXIT};

// 신용등급
enum {LEVEL_A=7, LEVEL_B=4, LEVEL_C=2};

// 계좌의 종류
enum {NORMAL=1, CREDIT=2};

#endif;
```

OOP 프로젝트 08단계
(AccountArray.h 추가)

494-498p

```
#ifndef __ACCOUN_ARRAY_H__
#define __ACCOUN_ARRAY_H__
#include "Account.h"
typedef Account * ACCOUNT_PTR;
class BoundCheckAccountPtrArray {
private:
    ACCOUNT_PTR * arr;
    int arrlen;

    BoundCheckAccountPtrArray(const BoundCheckAccountPtrArray& arr) { }
    BoundCheckAccountPtrArray& operator=(const BoundCheckAccountPtrArray& arr)
{ }

public:
    BoundCheckAccountPtrArray(int len=100);
    ACCOUNT_PTR& operator[] (int idx);
    ACCOUNT_PTR operator[] (int idx) const;
    int GetArrLen() const;
    ~BoundCheckAccountPtrArray();
};
#endif
```

494-498p

OOP 프로젝트 08단계
(AccountArray.cpp 추가)

```
#include "BankingCommonDecl.h"
#include "AccountArray.h"

BoundCheckAccountPtrArray::BoundCheckAccountPtrArray(int len) :arrlen(len)
{
	arr=new ACCOUNT_PTR[len];
}

ACCOUNT_PTR& BoundCheckAccountPtrArray::operator[] (int idx)
{
	if(idx<0 || idx>=arrlen)
	{
		cout<<"Array index out of bound exception"<<endl;
		exit(1);
	}
	return arr[idx];
}
```

494-498p

OOP 프로젝트 08단계
(AccountArray.cpp 추가)

```
ACCOUNT_PTR BoundCheckAccountPtrArray::operator[] (int idx) const
{
	if(idx<0 || idx>=arrlen)
	{
		cout<<"Array index out of bound exception"<<endl;
		exit(1);
	}
	return arr[idx];
}

int BoundCheckAccountPtrArray::GetArrLen() const
{
	return arrlen;
}

BoundCheckAccountPtrArray::~BoundCheckAccountPtrArray()
{
	delete []arr;
}
```

실습
string.cpp

C++ 기초
(string class)

- C/C++에는 문자열을 표현하는 기본 타입이 없다.
- Class와 연산자 overload 이용하여 **문자열을 기본 타입인 것처럼 사용하게 해 줌.**

```
#include <string>
#include <iostream>
using namespace std;
int main(void) {
        string s1, s2;
        cout << "비교할 문자열 2개 입력 :" << endl;
        cin >> s1 >> s2;
        cout << ((s1 == s2) ? "같다" : "다르다") << endl;
        return 0;
}
```

[실행결과]
비교할 문자열 2개 입력 :
문자열
문자열
같다

초록색 : 사용자가 입력한 부분.

실습
fileio.cpp

C++ 기초
(파일 입출력 방식)

["input.txt" 파일]
3
100
200
300

[실행결과]
100 200 300

```
#include <fstream>
#include <iostream>
using namespace std;
int main(void) {
        ifstream rFile("input.txt");
        if (rFile.is_open()) {
                int n;
                rFile >> n;
                int *pInt = new int[n];
                for (int i = 1; i <= n; i++)
                        rFile >> pInt[i];
                for (int i = 1; i <= n; i++)
                        cout << pInt[i] << " ";
                cout << endl;
                delete [] pInt;
        }
        rFile.close();
        return 0;
}
```

파일 내용 읽기.

가변 크기 버퍼 사용

C++ container class
(pair)

- **template <typename T1, typename T2> struct pair;**
- **데이터 요소의 2-튜플**
  - **고정된 순서에 따라 'first'(unique key), 'second'로 불림.**
  - **map 은 pair로 이루어진다.**
- **사용법**

  pair<type1, type2> p = make_pair(var1, var2);
- **비교 연산 가능 (== != < > <= >=), 정렬 가능**
  - **first값 기준. first가 같으면 second값 비교.**

```
#include <iostream>
using namespace std;
int main(void) {
        pair<int, string> p1 = make_pair(12, "kim");
        pair<int, string> p2 = make_pair(12, "kim");
        if (p1 == p2)
                cout << "same" << endl;
        else
                cout << "diff" << endl;
        return 0;
}
```

[실행결과]
same

C++ STL container
(deque : double ended queue)

시작과 끝에서 삽입/삭제

```
#include <iostream>
#include <deque>
using namespace std;
int main(void) {
        deque<pair<int, int>> _deque;
        pair<int, int> p1;
        _deque.push_back(make_pair(1, 10));
        _deque.push_back(make_pair(2, 20));
        _deque.push_back(make_pair(3, 30));
        _deque.push_back(make_pair(4, 40));
        p1 = _deque.back();
        _deque.pop_back();
        cout << "back:" << p1.first << "," << p1.second << endl;
        p1 = _deque.back();
        _deque.pop_back();
        cout << "back:" << p1.first << "," << p1.second << endl;
        while (!_deque.empty()) {
                p1 = _deque.front();
                _deque.pop_front();
                cout << "front:" << p1.first << "," << p1.second << endl;
        }
        return 0;
}
```

[실행결과]
back:4,40
back:3,30
front:1,10
front:2,20

back에 원소 추가

back 내용 확인 및 삭제. (2회)

front 내용 확인 및 삭제. (deque이 빌 때까지)