**객체지향프로그래밍 LAB #11**

**<기초문제>\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. 아래의 프로그램을 작성하시오. (/\*구현\*/ 부분을 채울 것, 표의 상단: 소스코드, 하단: 실행결과)

|  |
| --- |
| #include <iostream>  using namespace std;  class Point {  private:  int x;  int y;  static int numCreatedObjects;  public:  Point() : x(0), y(0) {  numCreatedObjects++;  }  // int \_x 와 int \_y를 입력으로 받는 생성자  /\*구현\*/  ~Point() {  cout << "Destructed..." << endl;  }  void setXY(int \_x, int \_y) {  //this-> 사용한 초기화  /\*구현\*/  }  int getX() const { return x; }  int getY() const { return y; }  // \*this + pt2 ->  Point operator+(Point& pt2) {  /\*구현\*/  }  //operator overloading(연산자 오버로딩)  Point& operator=(Point& pt) {  /\*구현\*/  }  static int getNumCreatedObject() { return numCreatedObjects; }  friend void print(const Point& pt);  friend ostream& operator<<(ostream& cout, Point& pt);  friend class SpyPoint;  };  //static 맴버 변수 초기화 (numCreatedObjects)  /\*구현\*/  //객체 call by reference시: const로 함수 입력시 const method만 함수에서 사용가능  // const: 객체 내부의 member data가 상수(변하지 않는다)  void print(/\*구현\*/) {  cout << pt.x << ", " << pt.y << endl;  }  //Point operator+(Point& pt1, Point& pt2){  // Point result(pt1.getX() + pt2.get(X), pt1.getY() + pt2.getY());  // return result;  //}  ostream& operator<<(ostream& cout, Point& pt) {  /\*구현\*/  }  class SpyPoint {  public:  //다음과 같이 출력 되도록 hack\_all\_info함수 구현  //Hacked by SpyPoint  //x: 40  //y: 60  //numCreatedObj.: 10  /\*구현\*/  };  int main() {  Point pt1(1, 2);  cout << "pt1 : ";  print(pt1);  cout << endl;  // 포인터  Point\* pPt1 = &pt1;  // pPt1의 값을 통해 getX, getY를 호출하여 출력  cout << "pt1 : ";  /\*구현\*/  // pPt1를 통해 호출 getX, getY를 호출하여 출력  cout << "pt1 : ";  /\*구현\*/  cout << endl;  //동적으로 Point\* pPt2할당하여 10,20넣은 뒤 ->사용하여 출력(pt1 출력 참고)  /\*구현\*/  cout << "pt2 : ";  /\*구현\*/  cout << endl;    //pPt1, pPt2의 메모리 해제  /\*구현\*/  cout << "pt1 NumCreatedObject : ";  cout << /\*구현\*/ << endl;  // 연산자 오버로딩  //pt4 = pt2, pt3값 더하기  /\*구현\*/  cout << "pt2 : ";  cout << pt2 << endl;  cout << "pt3 : ";  cout << pt3 << endl;  cout << "pt4 : ";  cout << pt4 << endl;  cout << "pt1 NumCreatedObject : ";  cout << /\*구현\*/ << endl << endl;  // object array  Point\* ptAry = /\*구현\*/;  cout << "pt2 NumCreatedObject : ";  cout << /\*구현\*/ << endl;  cout << endl;  // ptAry 메모리 해제  /\*구현\*/  cout << endl;  // friend class  SpyPoint spy;  cout << "pt1 info" << endl;  /\*구현\*/  cout << "pt4 info" << endl;  /\*구현\*/  return 0;  } |
|  |
| #include <iostream>  using namespace std;  class Point {  private:  int x;  int y;  static int numCreatedObjects;  public:  Point() : x(0), y(0) {  numCreatedObjects++;  }  // int \_x 와 int \_y를 입력으로 받는 생성자  Point(int \_x, int \_y) : x(\_x), y(\_y) {  numCreatedObjects++;  }  ~Point() {  cout << "Destructed..." << endl;  }  void setXY(int \_x, int \_y) {  //this-> 사용한 초기화  this->x = \_x;  this->y = \_y;  }  int getX() const { return x; }  int getY() const { return y; }  // \*this + pt2 ->  Point operator+(Point& pt2) {  Point result(this->x + pt2.x, this->y + pt2.y);  return result;  }  //operator overloading(연산자 오버로딩)  Point& operator=(Point& pt) {  this->x = pt.x;  this->y = pt.y;  return \*this;  }  static int getNumCreatedObject() { return numCreatedObjects; }  friend void print(const Point& pt);  friend ostream& operator<<(ostream& cout, Point& pt);  friend class SpyPoint;  };  //static 맴버 변수 초기화 (numCreatedObjects)  int Point::numCreatedObjects = 0;  //객체 call by reference시: const로 함수 입력시 const method만 함수에서 사용가능  // const: 객체 내부의 member data가 상수(변하지 않는다)  void print(const Point& pt) {  cout << pt.x << ", " << pt.y << endl;  }  //Point operator+(Point& pt1, Point& pt2){  // Point result(pt1.getX() + pt2.get(X), pt1.getY() + pt2.getY());  // return result;  //}  ostream& operator<<(ostream& cout, Point& pt) {  cout << pt.x << ", " << pt.y;  return cout;  }  class SpyPoint {  public:  //다음과 같이 출력 되도록 hack\_all\_info함수 구현  //Hacked by SpyPoint  //x: 40  //y: 60  //numCreatedObj.: 10  void hack\_all\_info(Point& pt) {  cout << "Hacked by SpyPoint" << endl;  cout << "x: " << pt.x << endl;  cout << "y: " << pt.y << endl;  cout << "numCreatedPbj : " << pt.numCreatedObjects << endl;  }  };  int main() {  Point pt1(1, 2);  cout << "pt1 : ";  print(pt1);  cout << endl;  // 포인터  Point\* pPt1 = &pt1;  // pPt1의 값을 통해 getX, getY를 호출하여 출력  cout << "pt1 : ";  cout << (\*pPt1).getX() << "," << (\*pPt1).getY() << endl;  // pPt1를 통해 호출 getX, getY를 호출하여 출력  cout << "pt1 : ";  cout << pPt1->getX() << "," << pPt1->getY() << endl;  cout << endl;  //동적으로 Point\* pPt2할당하여 10,20넣은 뒤 ->사용하여 출력(pt1 출력 참고)  Point\* pPt2 = new Point(10, 20);  cout << "pt2 : ";  cout << pPt2->getX() << ", " << pPt2->getY() << endl;  cout << endl;  //pPt1, pPt2의 메모리 해제  delete pPt2;  cout << "pt1 NumCreatedObject : ";  cout << pt1.getNumCreatedObject() << endl;  // 연산자 오버로딩  //pt4 = pt2, pt3값 더하기  Point pt2(10, 20), pt3(30, 40);  Point pt4 = pt2 + pt3;  cout << "pt2 : ";  cout << pt2 << endl;  cout << "pt3 : ";  cout << pt3 << endl;  cout << "pt4 : ";  cout << pt4 << endl;  cout << "pt1 NumCreatedObject : ";  cout << pt1.getNumCreatedObject() << endl << endl;  // object array  Point\* ptAry = new Point[5];  cout << "pt2 NumCreatedObject : ";  cout << pt2.getNumCreatedObject() << endl;  cout << endl;  // ptAry 메모리 해제  delete[] ptAry;  cout << endl;  // friend class  SpyPoint spy;  cout << "pt1 info" << endl;  spy.hack\_all\_info(pt1);  cout << "pt4 info" << endl;  spy.hack\_all\_info(pt4);  return 0;  } |

**텍스트이(가) 표시된 사진

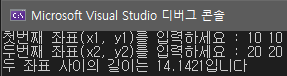
자동 생성된 설명<응용문제>\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. 좌표값 두 개를 입력받고 두 좌표 사이의 거리를 출력하시오.

* 좌표값은 초기화 시, 값을 따로 주지 않으면 x = 0, y = 0으로 초기화함.
* “좌표 - 좌표” 연산자를 Operator overloading을 활용하여 선언함.
  + 를 클래스 내에서 구현.
* “좌표 \* 좌표” 연산자를 Operator overloading을 활용하여 선언함.
  + 를 클래스 내에서 구현.
* 위 두 연산자를 활용하여 두 좌표 사이의 거리를 구함.
* 제곱근 사용을 위해 #include<cmath> 선언.

|  |
| --- |
| int main() {  int x1 = 0, y1 = 0, x2 = 0, y2 = 0;  Point\* pP1, \* pP2, \* pP3;  cout << "첫번째 좌표(x1, y1)를 입력하세요 : ";  cin >> x1 >> y1;  cout << "두번째 좌표(x2, y2)를 입력하세요 : ";  cin >> x2 >> y2;  pP1 = new Point(x1, y1);  pP2 = new Point(x2, y2);  pP3 = new Point(); //x,y가 0으로 초기화  /\* 아래의 방식으로도 x, y값 설정이 가능해야 함 \*/  //pP1->setPoint(x1, y1);  //pP1->setPoint(x2, y2);  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  \*pP3 = (\*pP1 - \*pP2) \* (\*pP1 - \*pP2);  /\* pP3을 활용하여 거리값을 구함 \*/  cout << "두 좌표 사이의 길이는 " << /\* 결과 값 \*/ << "입니다." << endl;  return 0;  } |

1-출력화면:



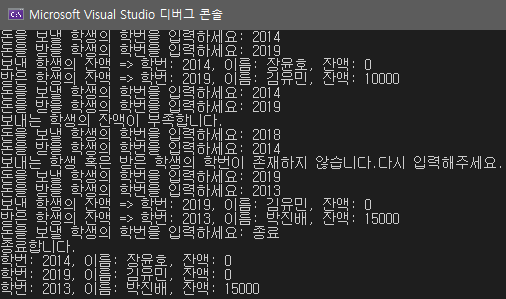
|  |
| --- |
| #include <iostream>  #include <cmath>  using namespace std;  class Point {  private:  int x, y;  public:  Point(int x = 0, int y = 0) : x(x), y(y) { }  void setPoint(int x, int y) { this->x = x; this->y = y; }  int getX() const { return x; }  int getY() const { return y; }  Point operator-(const Point pt) {  Point result(this->x - pt.x, this->y - pt.y);  return result;  }  Point operator\*(const Point pt) {  Point result(this->x \* pt.x, this->y \* pt.y);  return result;  }  Point& operator=(const Point& pt) {  this->x = pt.x;  this->y = pt.y;  return \*this;  }  };  int main() {  int x1 = 0, y1 = 0, x2 = 0, y2 = 0;  Point\* pP1, \* pP2, \* pP3;  cout << "첫번째 좌표(x1, y1)를 입력하세요: ";  cin >> x1 >> y1;  cout << "두번째 좌표(x2, y2)를 입력하세요: ";  cin >> x2 >> y2;  pP1 = new Point(x1, y1);  pP2 = new Point(x2, y2);  pP3 = new Point();  pP1->setPoint(x1, y1);  pP2->setPoint(x2, y2);  \*pP3 = (\*pP1 - \*pP2) \* (\*pP1 - \*pP2);  cout << "두 좌표 사이의 거리는 " << sqrt(pP3->getX() + pP3->getY()) << "입니다." << endl;  return 0;  } |
|  |

2. 학생들끼리 송금하는 프로그램을 작성하시오.

* 한번 송금할 때, 돈을 보내는 학생의 전 재산(balance)를 송금함.
* Account(계좌) class는 멤버변수로 string name(이름), string id(학번), int balance를 갖고 있도록 함.
* 사용자로부터 돈을 보낼 학생의 학번과 돈을 받을 학생의 학번을 입력 받음. 이때, 다음의 경우에 대해서는 다시 입력 받도록 함.
  + 돈을 보낼 학생과 돈을 받을 학생의 학번이 동일한 경우
  + 보낼 학생 혹은 받을 학생의 학번이 존재하지 않는 경우
  + 보낼 학생의 잔액이 0인 경우
* setBalance함수를 사용하지 않고, Operator overloading(+, -)를 이용하여 송금 후 보낸 학생의 잔액과 받은 학생의 잔액을 계산함.
* 송금이 완료된 후, Operator overloading(<<)을 이용하여 돈을 보낸 학생과 받은 학생의 계좌를 출력함.
* 사용자로부터 돈을 보낼 학생의 학번을 “종료”라고 입력받았을 경우 Operator overloading(<<)을 이용하여 모든 학생의 계좌를 출력하고, 프로그램을 종료함.

|  |
| --- |
| int main() {  Account acnt[3] = {  Account("장윤호", "2014", 10000),  Account("김유민", "2019", 0),  Account("박진배", "2013", 5000),  };  /\* 구현 \*/  } |

2-출력화면:



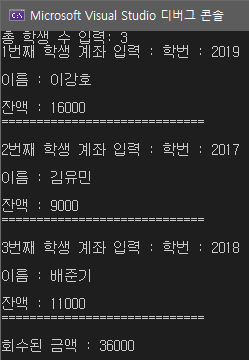
|  |
| --- |
| #include <iostream>  using namespace std;  class Account {  private:  string name, id;  int balance;  public:  Account(string name, string id, int balance) : name(name), id(id), balance(balance) { }  string getID() {  return id;  }  int getBalance() {  return balance;  }  Account operator+(Account& acc) {  Account result(this->name, this->id, this->balance + acc.balance);  return result;  }  Account operator-(Account& acc) {  Account result(this->name, this->id, this->balance - acc.balance);  return result;  }  friend ostream& operator<<(ostream& cout, const Account& acc);  };  ostream& operator<<(ostream& cout, const Account& acc) {  cout << "학번: " << acc.id << ", 이름: " << acc.name << ", 잔액: " << acc.balance;  return cout;  }  int main() {  Account acc[3] = {  Account("장윤호", "2014", 10000),  Account("김유진", "2019", 0),  Account("박진배", "2013", 5000)  };  string withdrawID;  string depositID;  int withdraw\_idx = -1;  int deposit\_idx = -1;  while (true) {  withdraw\_idx = -1;  deposit\_idx = -1;  cout << "돈을 보낼 학생의 학번을 입력하세요: ";  cin >> withdrawID;  if (withdrawID == "종료") {  cout << "종료합니다." << endl;  for (int i = 0; i < 3; i++)  cout << acc[i] << endl;  break;  }  cout << "돈을 받을 학생의 학번을 입력하세요: ";  cin >> depositID;  for (int i = 0; i < 3; i++)  if (withdrawID == acc[i].getID())  withdraw\_idx = i;  for (int i = 0; i < 3; i++)  if (depositID == acc[i].getID())  deposit\_idx = i;  if (withdrawID == depositID) {  cout << "돈을 보낼 학생의 학번과 돈을 받을 학생의 학번이 동일합니다." << endl;  continue;  }  else if (withdraw\_idx == -1 || deposit\_idx == -1) {  cout << "보내는 학생 혹은 받는 학생의 학번이 존재하지 않습니다. 다시 입력해주세요." << endl;  continue;  }  else if (acc[withdraw\_idx].getBalance() == 0) {  cout << "보내는 학생의 잔액이 부족합니다." << endl;  continue;  }  else {  acc[deposit\_idx] = acc[deposit\_idx] + acc[withdraw\_idx];  acc[withdraw\_idx] = acc[withdraw\_idx] - acc[withdraw\_idx];  cout << "보낸 학생의 잔액 => " << acc[withdraw\_idx] << endl;  cout << "받은 학생의 잔액 => " << acc[deposit\_idx] << endl;  continue;  }  }  return 0;  } |
|  |

3. 학생 계좌 정보를 입력받고 학생 계좌 정보들을 모두 삭제하여 삭제된 계좌들의 잔액 총합을 출력하는 프로그램을 작성하시오.

* 총 학생 수를 사용자로부터 입력받음.
* 학생 계좌 정보 Account class는 학번, 이름, 잔액을 멤버 변수로 가지고 있음.
* Account 클래스 배열을 동적으로 생성함.
* 각 학생의 학번, 이름, 잔액을 입력받음. 이 때, 중복된 학번을 입력받으면 프로그램을 종료함.
* 모두 입력 받은 후, Account 클래스 배열을 delete하여 모든 학생 계좌의 잔액 총합을 출력함.

Hint) Account class 내 멤버 변수에서 static 변수를 사용, Account 클래스의 소멸자를 적절히 이용.

3-출력화면:



|  |
| --- |
| #include <iostream>  using namespace std;  class Account {  private:  string id, name;  int balance;  static int sum;  public:  Account(string id = "", string name = "", int balance = 0) : id(id), name(name), balance(balance) { }  ~Account() {  sum += balance;  }  void setAcc(string id, string name, int balance) {  this->id = id;  this->name = name;  this->balance = balance;  }  string getID() {  return id;  }  static int getSum() {  return sum;  }  };  int Account::sum = 0;  int main() {  int accNum = 0;  cout << "총 학생 수 입력: ";  cin >> accNum;  Account\* acc = new Account[accNum];  string ID;  string name;  int balance;  for (int i = 0; i < accNum; i++) {  cout << i + 1 << "번째 학생 계좌 입력 : " << "학번: ";  cin >> ID;  for (int j = 0; j < i; j++)  if (acc[j].getID() == ID)  exit(100);  cout << endl;  cout << "이름: ";  cin >> name;  cout << endl;  cout << "잔액: ";  cin >> balance;  acc[i].setAcc(ID, name, balance);  cout << "==========================" << endl << endl;  }  delete[] acc;  cout << "회수된 금액: " << Account::getSum() << endl;  return 0;  } |
|  |