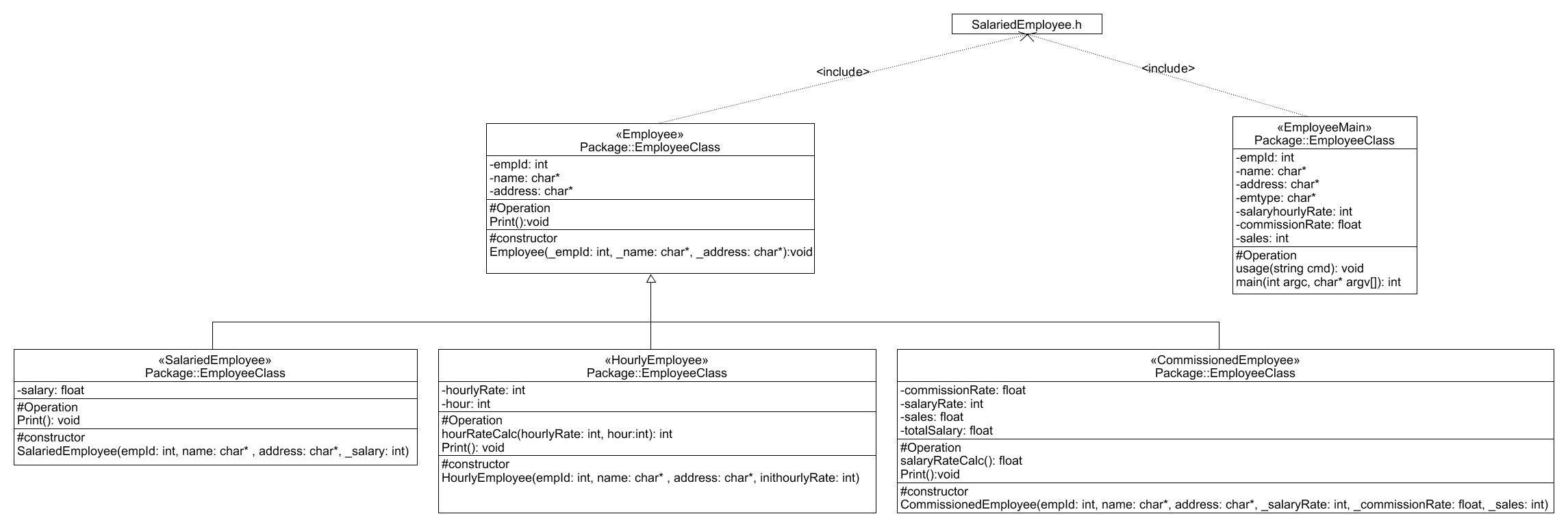
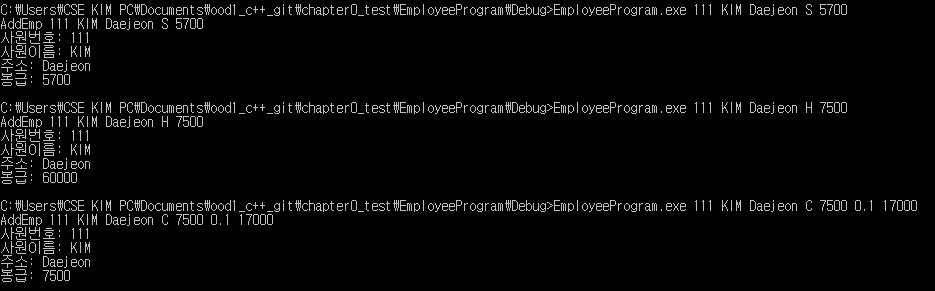
**[전반적인 UML설계]**



**[실습 문제 1]**



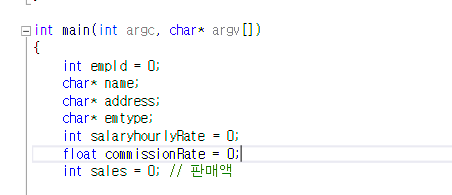
**< 직원별 유형을 추가하여 정보를 출력하는 C++ 프로그램 작성 결과 화면 >**

▹ 판매원의 경우는 sales라는 판매액을 추가적으로 생성하여 작성하였습니다.

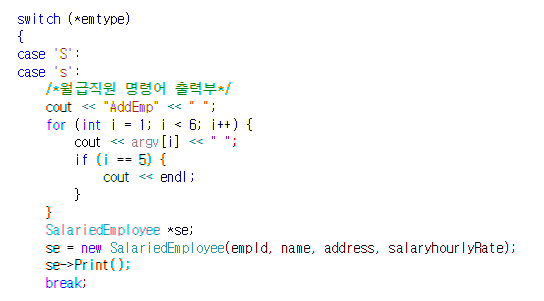
**[실습문제1 코드설계]**

<EmployeeMain.cpp> 하나에서 출력을 할 수 있게 코드를 설계 하였습니다.

기본 설계는 Main부분에 아래와 같이 argc, argv 배열을 사용하여 cmd창으로부터 기존에 원하는 형태로 명령어를 받아 emptype에서 각 상황에 맞게 switch문을 통하여 각 분기를 나누었습니다. 이후 swich문을 통하여 나누어진 각각의 case에서 강의자료에서 요하는대로 문구를 출력하게끔 진행을 하였습니다.

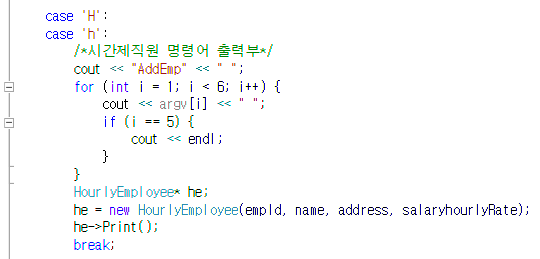


**<Main에서 argc와 argv[] 파라미터부분>**



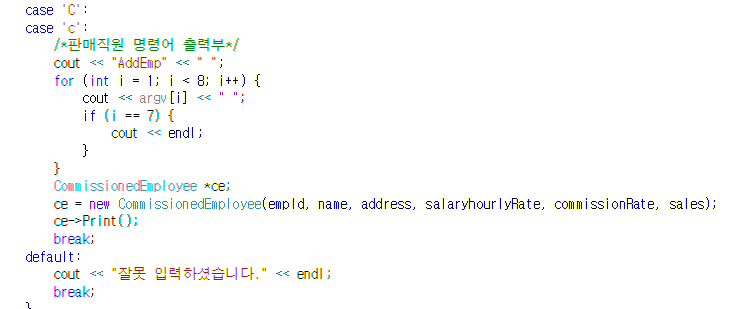
**<Switch문을 이용하여 S|H|C 형태로 받아서 S(월급제 직원)의 경우 구현코드>**

S혹은 s로 대소문자 모두 받을 수 있게 구현을 하였으며, 각 부분에서 AddEmp empId, name, address, Salary를 표시하게끔 하는 부분입니다. 출력부는 각 argv 배열을 순차적으로 출력을 하게끔 진행을 하였으며 위의 순서대로 출력이 됩니다. for문을 사용하여 위와 같이 구현하였습니다.



**<Switch문을 이용하여 S|H|C 형태로 받아서 H(시간제 직원)의 경우 구현코드>**

코드 구현은 위의 월급제 직원 부분 출력하는 부분과 동일합니다.



**<Switch문을 이용하여 S|H|C 형태로 받아서 C(판매원)의 경우 구현코드>**

구현 코드는 위의 월급제 직원의 명령어 출력부와 같은 방식으로 구현되었으며 추가적으로 월급, 판매수수료, 판매액이 추가되어 위와 같이 argv로 출력할 부분의 수가 늘어난 형태입니다.

또한 이 부분에서 기존의 요구사항에 나와있지 않은 판매액을 받을 수 있게끔 구현을 하였습니다.

**[실습 문제2 및 심화]**

**[설계 아이디어]**

위의 UML을 토대로 class 및 변수, 함수들을 구현하였습니다.

C++의 객체지향적 설계 방식을 적용하여 class의 전반적인 상속 관계, Employee 프로그램안에 들어있는 내부 클래스에 대한 정의, 각 클래스 속의 메소드와 변수 등을 <SalariedEmployee.h>에 정의를 하였습니다. <SalariedEmployee.h> 헤더 파일에 전반적인 프로그램의 뼈대, 설계도를 구현하였고 이를 include받아서 각 <Employee.cpp>, <EmployeeMain.cpp>에 include를 하였습니다. 이후 전반적인 구현은 각 클래스의 cpp파일에 코드를 구현하여, 실질적인 함수의 구현, 변수 초기화, 생성자 등을 이 부분에 구현하였습니다.

Employee의 하위 클래스인 <SalariedEmployee>, <HourlyEmployee>, <CommissionedEmployee>에도 각각 <SalariedEmployee.h>헤더파일을 include시켜서 헤더파일의 설계를 기반으로 cpp파일에 실질적 구현을 하는 형태의 C++의 객체지향적 설계를 하였습니다.

**[코드 구현 및 설명]**

|  |
| --- |
| /\*전반적인 Employee프로그램의 헤더 구성\*/  #ifndef \_\_SALARIEDEMPLOYEE\_H\_  #define \_\_SALARIEDEMPLOYEE\_H\_  //직원 클래스  class Employee  {  private:  int empId;  char\* name;  char\* address;  public:  Employee(int \_empId, char\* \_name, char\* \_address);  void Print();  };  // 시간제 직원 클래스  class HourlyEmployee : public Employee  {  private:  int hourlyRate;  int hour = 8; // 오전 9시~ 저녁 6시 기준  public:  HourlyEmployee(int empId, char\* name, char\* address, int hourlyRate);  int hourRateCalc(int hourlyRate, int hour);  void Print();  };  // 월급제 직원 클래스  class SalariedEmployee : public Employee  {  private:  float salary;  public:  SalariedEmployee(int empId, char\* name, char\* address, int \_salary);  void Print();  };  // 판매 수수료 직원  class CommissionedEmployee : public Employee  {  private:  float commissionRate; //수수료  int salaryRate; //월급  float sales; // 판매액  float totalSalary; // 총월급  public:  CommissionedEmployee(int empId, char\* name, char\* address, int \_salaryRate, float \_commissionRate, int \_sales);  float salaryRateCalc();  void Print();  };  #endif |

**< SalariedEmployee.h 의 코드 구현>**

전반적인 헤더파일의 구현은 최상위인 <Employee 클래스>, <SalariedEmployee 클래스>, <HourlyEmployee 클래스>, <CommissionedEmployee 클래스>를 각각 써서 정의하였고 각각의 변수명은 각 class에서만 사용될 것이기에 private로 정의하여 변수를 설정하였습니다. 각 class에서 사용되는 operation들은 public으로하여 외부에서도 사용 가능하게끔 구현하였습니다.

Employee 클래스의 하위 클래스인 <SalariedEmployee 클래스>, <HourlyEmployee 클래스>, <CommissionedEmployee 클래스>는 각각 “: pulic Employee”를 사용하여 Employee 클래스를 상속받게 구현하였습니다.

각 class의 생성자는 pulic으로 명기하였고 실질적인 생성자가 할당 받는 부분들은 모두 각 클래스의 cpp파일에 정의하였습니다.

* 각 클래스별 세부적인 변수명 각 변수 타입, 연산자명은 위의 UML도식에 모두 명기되어 있습니다.

|  |
| --- |
| /\*201204441 김수현 상위 Employee 파일 구현\*/  #include<iostream>  #include<iomanip>  #include<cstring>  #include "SalariedEmployee.h"  using namespace std;  Employee::Employee(int \_empId, char\* \_name, char\* \_address) : empId(\_empId) {  name = new char[sizeof(\_name) + 1];  address = new char[sizeof(\_address) + 1];  strcpy(name, \_name);  strcpy(address, \_address);  }  void Employee::Print()  {  cout << "사원번호: " << empId << endl;  cout << "사원이름: " << name << endl;  cout << "주소: " << address << endl;  } |

**< Employee.cpp의 코드 구현>**

상위 클래스인 Employee클래스의 구현부인 Employee.cpp파일 부분이다. 클래스다이어그램에 나와 있듯, 생성자부분과 Print()연산자 2가지로 구현이 되어있다.

상위, 하위 클래스에 있는 변수 3가지 empId, name, address의 변수의 값을 초기화하는 생성자 부분인 Employee::Employee(int \_empId, char\* \_name, char\* \_address) : empId(\_empId) 부분을 보면 EmployeeMain으로부터 받아온 argv[1], argv[2], argv[3] 부분 중 문자열을 포인터로 받는 name과 address에 대한 것을 정의하였다. 각 \_name, \_address의 문자열의 최종 사이즈 + 1(‘\n’)까지 포함하여 받아온다. Char는 1개의 문자씩을 받아오기 때문에 위와 같이 new char[sizeof(\_name) + 1]을 하여 “KIM\n”을 받아와 name이라는 변수에 저장하게 한 것이다. 주소인 address변수 또한 이와 같은 방식으로 변수에 저장이 된다.

Strcpy를 통하여 각 변수에 문자열을 저장하여 다른 하위 클래스에 사용하게끔 다음과 같이 정의를 하였다.

Print()함수의 구현은 위와 같이 미리 정의된 변수 empId, name, address에 값을 받아와 그대로 출력하는 형태로 구현하였다.

|  |
| --- |
| SalariedEmployee::SalariedEmployee(int empId, char\* name, char\* address, int \_salary) : Employee(empId, name, address), salary(\_salary)  {}  void SalariedEmployee::Print()  {  Employee::Print();  cout << "봉급: " << salary << endl;  } |

**< SalariedEmployee.cpp의 코드 구현>**

SalariedEmployee.cpp의 구현부분으로 SalariedEmployee의 실제 구현부분이다. SalariedEmployee::SalariedEmployee(int empId, char\* name, char\* address, int \_salary) : Employee(empId, name, address), salary(\_salary) 부분을 통하여 SalariedEmployee에서 사용할 변수들을 모두 정의하고 있음을 볼 수 있다. 앞서 empId, name, address는 Employee라는 상위 객체로부터 받아오는것을 “: Employee(empId,name,address)”를 통하여 정의하였다. 각 파라미터로부터 값을 받아오고 받아온 값들중 앞에 3개(empId, name, address)는 상위 클래스인 Employee 클래스의 생성자 부분으로 가서 해당 변수들을 받아온다. Salray의 경우 private으로 정의 되어있는 클래스 자체내 변수에 \_salary의 값을 받아오는 형태로 변수들을 모두 초기화하였다.

Print()메소드의 경우 상위 Employee클래스의 Print()함수를 override(재정의)를 하여 만들었다. 상위 class의 Print()를 호출하여 불러오고 추가적으로 월급제직원의 봉급을 추가적으로 출력하여 최종적인 월급 출력을 하게 하였다.

월급제직원의 경우 받은 월급을 계산 없이 그대로 표현하는 것이기에 위와 같이 Print() 메소드 하나만 구현하였다.

|  |
| --- |
| HourlyEmployee::HourlyEmployee(int empId, char\* name, char\* address, int inithourlyRate) :Employee(empId, name, address), hourlyRate(inithourlyRate)  {  hourlyRate = HourlyEmployee::hourRateCalc(inithourlyRate, hour);  }  int HourlyEmployee::hourRateCalc(int hourlyRate, int hour)  {  int result;  result = hourlyRate \* hour;  return result;  }  void HourlyEmployee::Print()  {  Employee::Print();  cout << "봉급: "<< HourlyEmployee::hourlyRate << endl;  } |

**< HourlyEmployee.cpp의 코드 구현>**

HourlyEmployee는 다음과 같이 생성자 1개와 2개의 메소드로 구현이 되었다.

맨 위의 HourlyEmployee::HourlyEmployee() 생성자에서는 3개(empId, name, address)의 변수는 상위 클래스인 Employee에서 받는 형태로 진행하였다. hourlyRate에는 시급 계산식인 시급 \* 시간을 하여 계산한 값을 집어 넣어야하므로, 생성자에서 hourlyRate는 아래의 hourRateCalc에서 계산하여 반환된 값을 저장하게 하였다.

**(심화문제)** hourRateCalc()메소드에서 hour에는 오전 9시 ~ 저녁 6시 기준(8시간 근로)를 기준으로 8값을 default값으로 저장하여 계산을 수행한다. 따라서 실질적으로 hourlyRate(시급)을 받아서 입력받은 시급 \* 8을 최종 시급으로 계산하여 result변수에 저장하여 반환하게 된다.

Print()메소드에서는 hourRateCalc()에서 반환한 값을 받아서 Employee함수에서 Print()함수를 불러 앞서 직원의 기본 정보를 모두 표기하고 마지막에 계산된 총 시급을 출력하게 하였다.

|  |
| --- |
| CommissionedEmployee::CommissionedEmployee(int empId, char\* name, char\* address, int \_salaryRate, float \_commissionRate, int \_sales) :Employee(empId, name, address),commissionRate(\_commissionRate),salaryRate(\_salaryRate),sales(\_sales)  {  totalSalary = CommissionedEmployee::salaryRateCalc();  }  float CommissionedEmployee::salaryRateCalc()  {  float result = 0.0;  result = salaryRate + (sales \* commissionRate);  return result;  }  void CommissionedEmployee::Print()  {  Employee::Print();  cout << "봉급: " << totalSalary << endl;  } |

**< CommissionEmployee.cpp의 코드 구현>**

CommissionEmployee에서도 마찬가지로 Employee를 상속받으며 1개의 생성자와 2개의 메소드로 구성되어있다.

CommissionEmployee::CommissionedEmployee()는 생성자로 앞서 위의 2개의 하위클래스와 마찬가지로 상위 클래스인 Employee클래스로부터 3개의 변수(empId, name, address)를 상속받아 저장하며, \_salaryRate, \_commissionRate, \_sales의 값들을 private로 정의한 클래스 변수salaryRate, commissionRate, sales에 각각 저장한다. 또한 클래스 내부에 정의된 totalSalary변수에다가 클래스내 봉급계산을 수행하는 salaryRateCalc()로부터 계산된 월급 값을 저장한다.

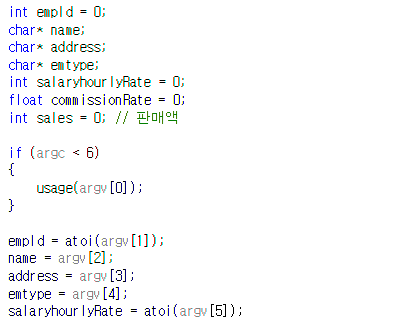
**(심화 문제)** 월급을 계산하는 salaryRateCalc()메소드에서는 클래스 내부 변수에 저장된 값들을 받아서 월급(salaryRate) + (판매액(sales) \* 수수료(commissionRate))를 계산하여 result변수에 값으로 저장하며 이를 반환하여 최종적으로 생성자를 통하여 클래스 변수 totalSalary에 저장하는 일을 수행한다.

Print()메소드에서는 상위 클래스인 Employee의 print()를 불러와 기본적인 직원의 정보 출력 외에 추가적으로 계산된 판매원의 봉급(totalSalary)을 출력하는 기능을 수행한다.

|  |
| --- |
| void usage(string cmd)  {  cout << "usage:" << cmd << " <EmpId> <name> <address> S|H|C" << "<salary|hourlyRate> [commisionRate]" << endl;  exit(0);  }  int main(int argc, char\* argv[])  {  int empId = 0;  char\* name;  char\* address;  char\* emtype;  int salaryhourlyRate = 0;  float commissionRate = 0;  int sales = 0; // 판매액  if (argc < 6)  {  usage(argv[0]);  }    empId = atoi(argv[1]);  name = argv[2];  address = argv[3];  emtype = argv[4];  salaryhourlyRate = atoi(argv[5]);  /\*commissionRate & sales (수수료 부분, 판매액 부분에 대한 초기화)\*/  if (commissionRate == 0)  {  commissionRate = 0;  }  else {  commissionRate = atof(argv[6]);  }  if (sales == 0)  {  sales = 0;  }else {  sales = atof(argv[7]);  }  switch (\*emtype)  {  case 'S':  case 's':  /\*월급직원 명령어 출력부\*/  cout << "AddEmp" << " ";  for (int i = 1; i < 6; i++) {  cout << argv[i] << " ";  if (i == 5) {  cout << endl;  }  }  SalariedEmployee \*se;  se = new SalariedEmployee(empId, name, address, salaryhourlyRate);  se->Print();  break;    case 'H':  case 'h':  /\*시간제직원 명령어 출력부\*/  cout << "AddEmp" << " ";  for (int i = 1; i < 6; i++) {  cout << argv[i] << " ";  if (i == 5) {  cout << endl;  }  }  HourlyEmployee\* he;  he = new HourlyEmployee(empId, name, address, salaryhourlyRate);  he->Print();  break;    case 'C':  case 'c':  /\*판매직원 명령어 출력부\*/  cout << "AddEmp" << " ";  for (int i = 1; i < 8; i++) {  cout << argv[i] << " ";  if (i == 7) {  cout << endl;  }  }  CommissionedEmployee \*ce;  ce = new CommissionedEmployee(empId, name, address, salaryhourlyRate, commissionRate, sales);  ce->Print();  break;  default:  cout << "잘못 입력하셨습니다." << endl;  break;  }  } |

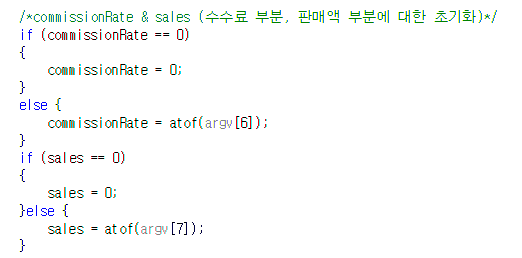
**< EmployeeMain.cpp의 코드 구현>**

EmployeeMain클래스에서 프로그램 내의 클래스들을 활용하여 실제 프로그램을 연결하여 작동 할 수 있도록 구성하였다. 메인 메소드의 핵심적인 부분은 switch문을 통하여 동작하는 것이 핵심으로, argc, argv[] 배열을 파라미터로 받아서 각각을 받아온다.

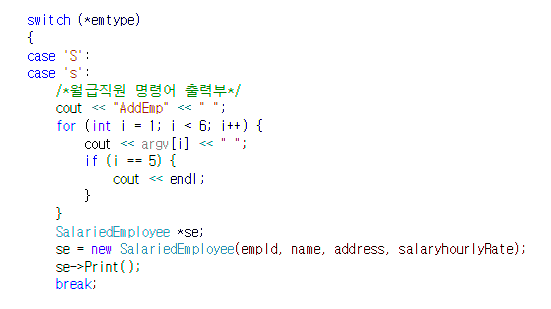


Main() 메소드에서 실제 필요한 변수들을 각각 정의하고 하위에 각각 argv 배열의 순서대로 매칭시켜서 변수에 저장한다. 실제 받는것도 해당 변수에 저장되는 순서대로 입력해야한다.

Usage()메소드에서는 해당부분이 salaryhourlyRate까지 즉, argv[5]까지 5개의 변수를 받지 못했을시에 출력되는 문장으로 문장의 출력은 실행 파일명(argv[0]), 사원번호 EmpId(argv[1]), 사원명name(argv[2]), 주소address(argv[3]), 사원타입 emtype(argv[4]) , 사원 월급(시급) salaryhourlyRate  
(argv[5])까지 받지 않으면 이를 쓰라고 출력하고 프로그램이 종료되게 함수를 구현하였다.

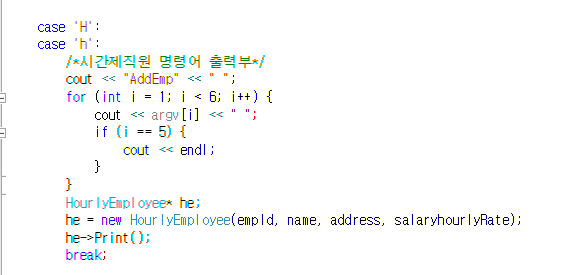


이후 emtype가 C(판매원)의 경우에는 salaryEmployee보다 파라미터가 2종류 가량 추가가 된다. CommissionRate(수수료)와 sales(판매액)이 추가가 되며, 다른 2종류의 사원들은 이를 받지 않기 때문에 해당 경우를 고려하여 이와 같이 적용하였다. C(판매원)가 아닌 경우 앞서 새로 추가된 commissionRate와 sales가 0인 경우가 그대로 내려올 것이므로 한번 더 0으로 초기화 하였고, 0이 아닌 경우는 각각에 cmd창으로부터 받은 수수료와 판매액을 각각의 변수에 저장하게 하였다.

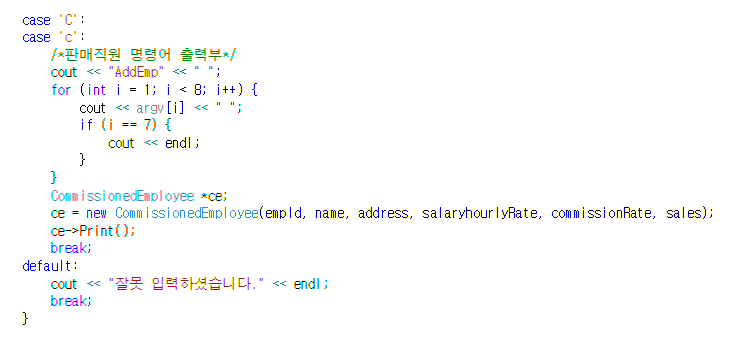


**<월급제 직원의 경우 Switch문 코드 구현>**

이후 각 사원의 종류에 따라 switch문을 통하여 emtype에 따라 S|H|C인 경우를 대소문자 구분없이 받아서 처리하게 구현하였다. 출력 부분은 앞서 설명하였으므로 밑부분에 SalariedEmployee의 객체를 생성한 후 해당 객체의 Print()메소드를 불러오게 진행하였다 이는 월급제 직원뿐 아니라 다른 종류의 직원들의 경우도 모두 마찬가지로 구현하였다.



**<시간제 직원의 경우 Switch문 코드 구현>**

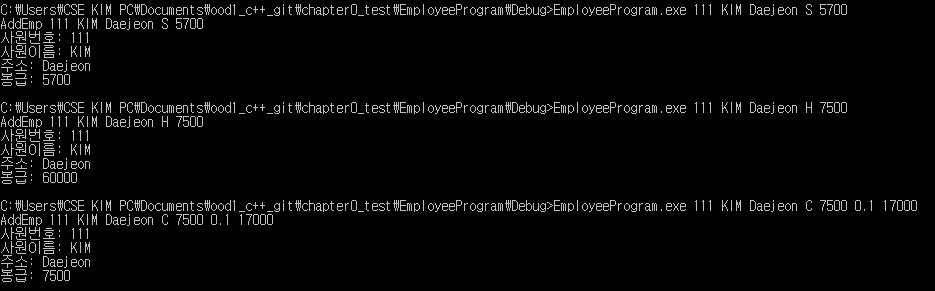


**<판매원의 경우 및 그 외의 경우 switch문 코드 구현>**

혹 S|H|C의 경우 외에 다른 문자가 왔을 시에는 잘못입력하였다는 문구를 출력하게끔 하였다.

그 이후의 동작들은 모두 각 클래스에서 구현된 방식으로 동작하여 최종적으로 직원의 기본 정보 및 봉급을 계산하여 출력할 수 있게 구현하였다.

**[실습 문제2 및 심화문제의 결과]**



**<실습문제2 & 심화문제의 결과화면>**

위의 설계대로 정확히 동작하며 제대로 구현이 되어 출력되는 모습을 볼 수 있다.

**[느낀점]**

C++에 대한 기본적인 내용을 지나서 상당히 심화된 내용까지 한꺼번에 알아야하는 것으로 인하여 많은 시간이 걸렸고 이를 토대로 C++에 대해 잠시나마 이해할 수 있는 계기가 되었다고 판단된다. 독특했던 점들은 pointer를 다른 객체지향언어와 달리 C와 마찬가지로 사용을 하며 이로 인한 객체 생성 및 메소드 호출 등에서 애먹은 점이 있었지만 조교님께 질문과 인터넷, 책을 토대토 찾아본 것을 통하여 익히게 되었다. 이 부분은 추가적으로 더 공부해볼 예정이다. 이후 controller와 main class의 동작을 한번 더 세분화하여 나누는 작업을 추가적으로 시간이 허락한다면 진행해보고자 한다.

위의 소스코드 및 보고서의 파일들은 Github에 게재하였다.

최종 파일에 대한 github주소: https://github.com/skyfall13/OOD00\_Employee.git