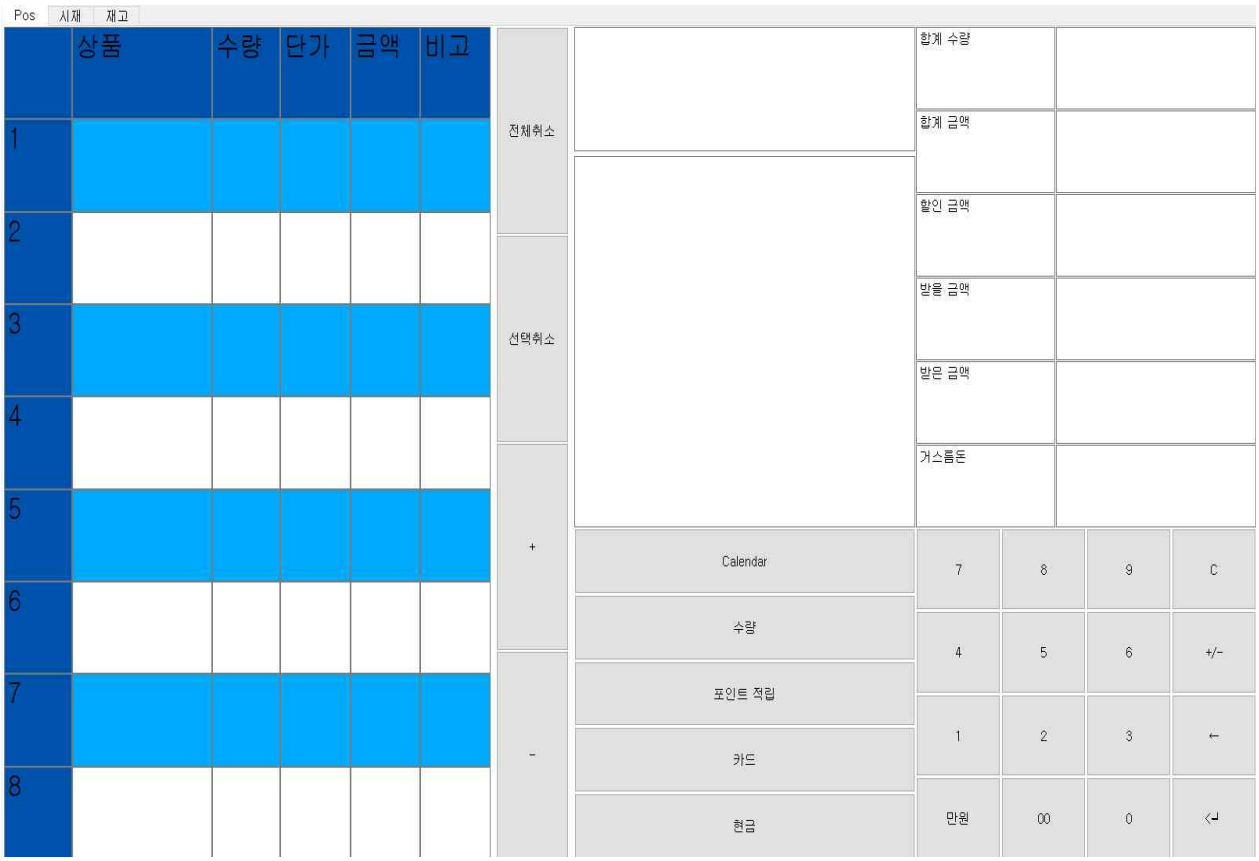
<Project#4 : Market Managing System>



TEAM NAME: 오지는 객체지향 9조

Leader 20185168 이기태

20183622 김태우

20185785 이상윤

Presenter 20184165 이태운

20185949 최예슬

목차

1. 설계 방향

실제 마트에서 사용하고 있는 포스기와 같이 현실성있고 편의성을 강조해 사용하기 쉬운 포스기를 만드는데 가장 큰 의의를 두었다.

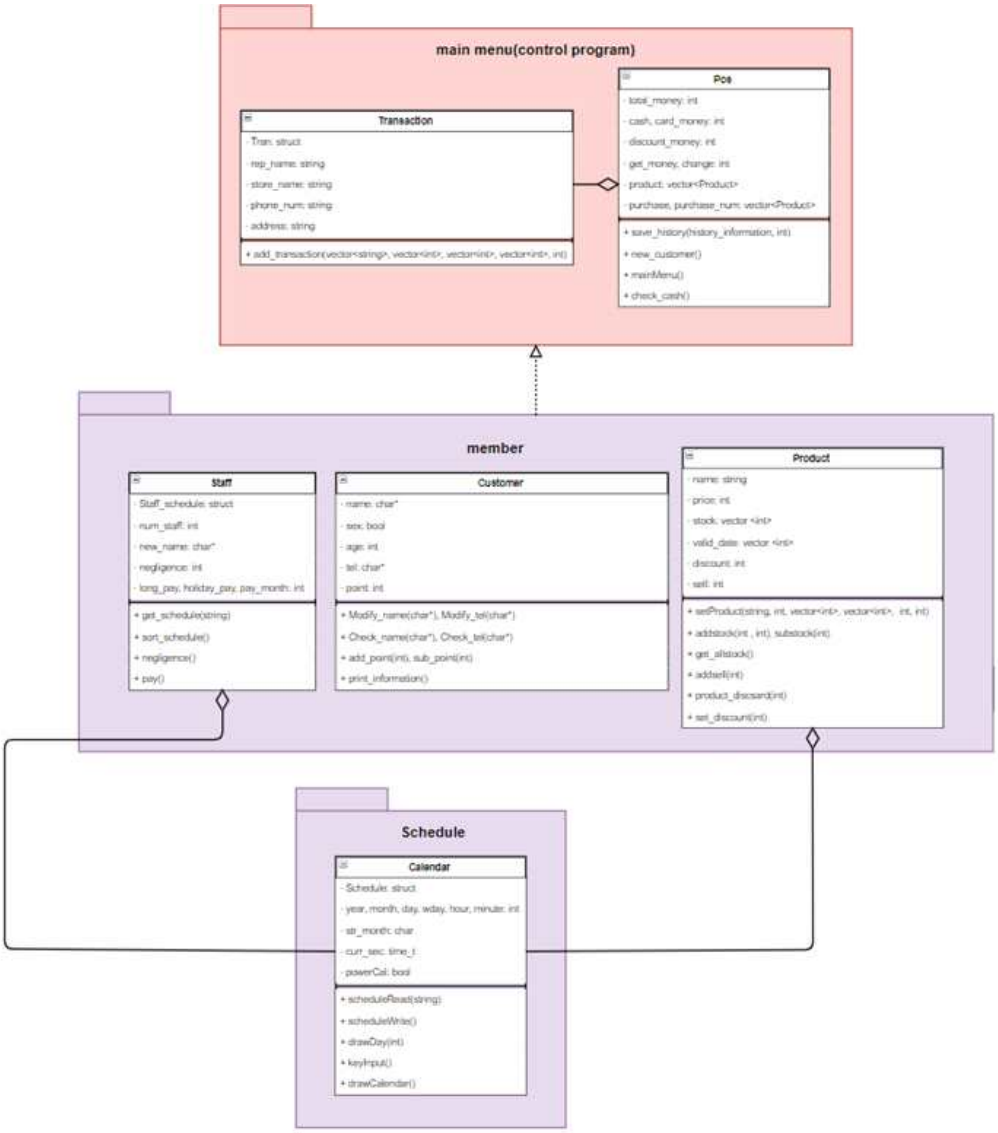
현실성있는 포스기를 구현하기 위해 비주얼 스튜디오 프로그램으로는 한계를 느끼고 qt라는 GUI툴을 통해 값을 받아들이고 UI를 구성하였다.

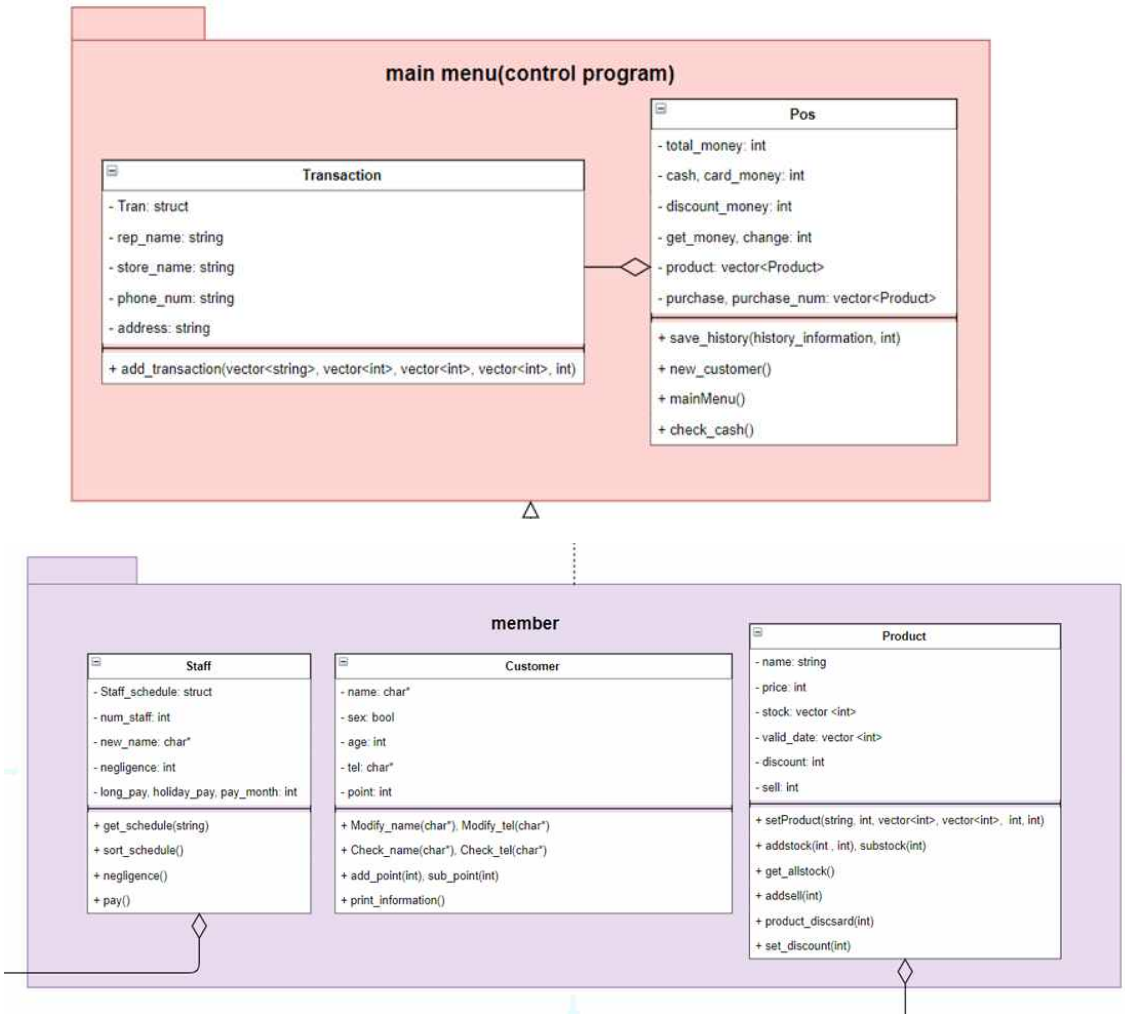
고객 / 제품 / 직원 / 포스기 / 시간 / 거래내역으로 클래스를 나누어 최대한 많은 기능을 구현하는데 주력했다. 실제 포스기처럼 거스름돈 계산, 포인트 적립, 카드 / 현금 계산, 거래내역 / 영수증 출력 등 현실성있는 포스기의 기능들을 구현하는데 성공했고 처음보는 사람이더라도 쉽게 사용할 수 있게해 편의성을 충족시켰다. 또 실제 포스기와는 차별을 두어 특색있는 기능 또한 구현하였다.

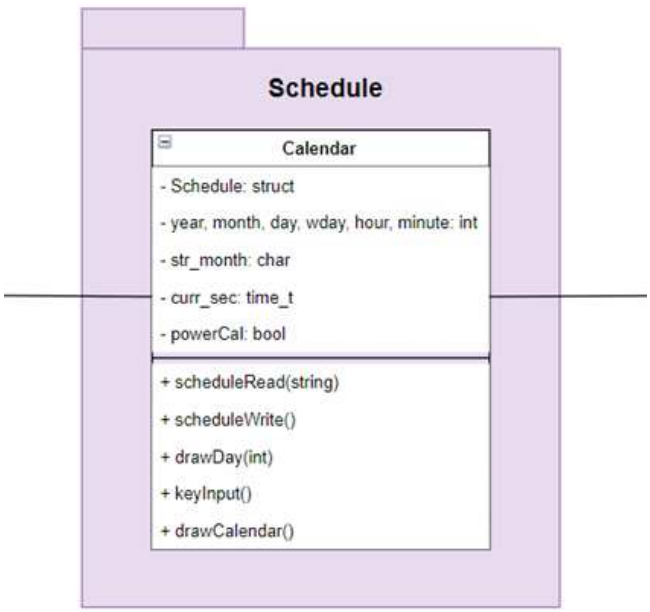
이처럼 사용하기 쉽고 현실성있게 개발하는 쪽으로 설계방향을 잡았다.

2. UML

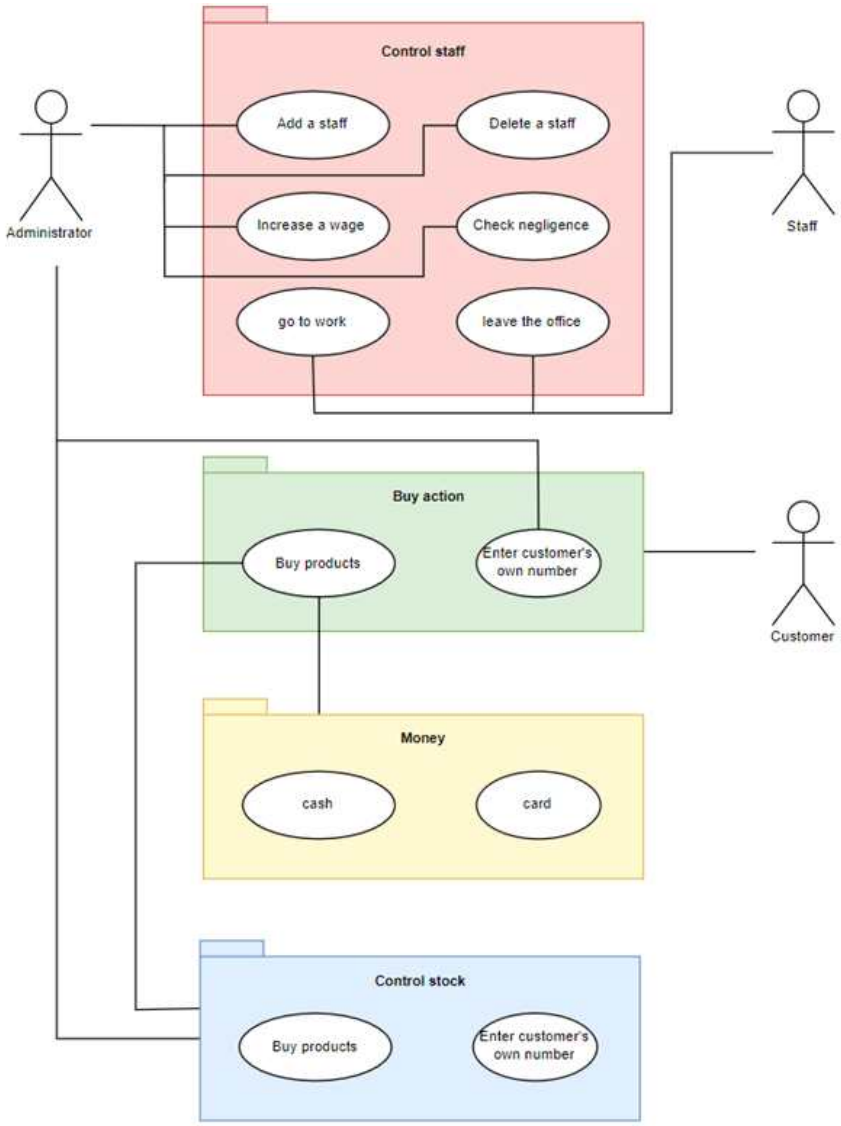
(1) Class Diagram



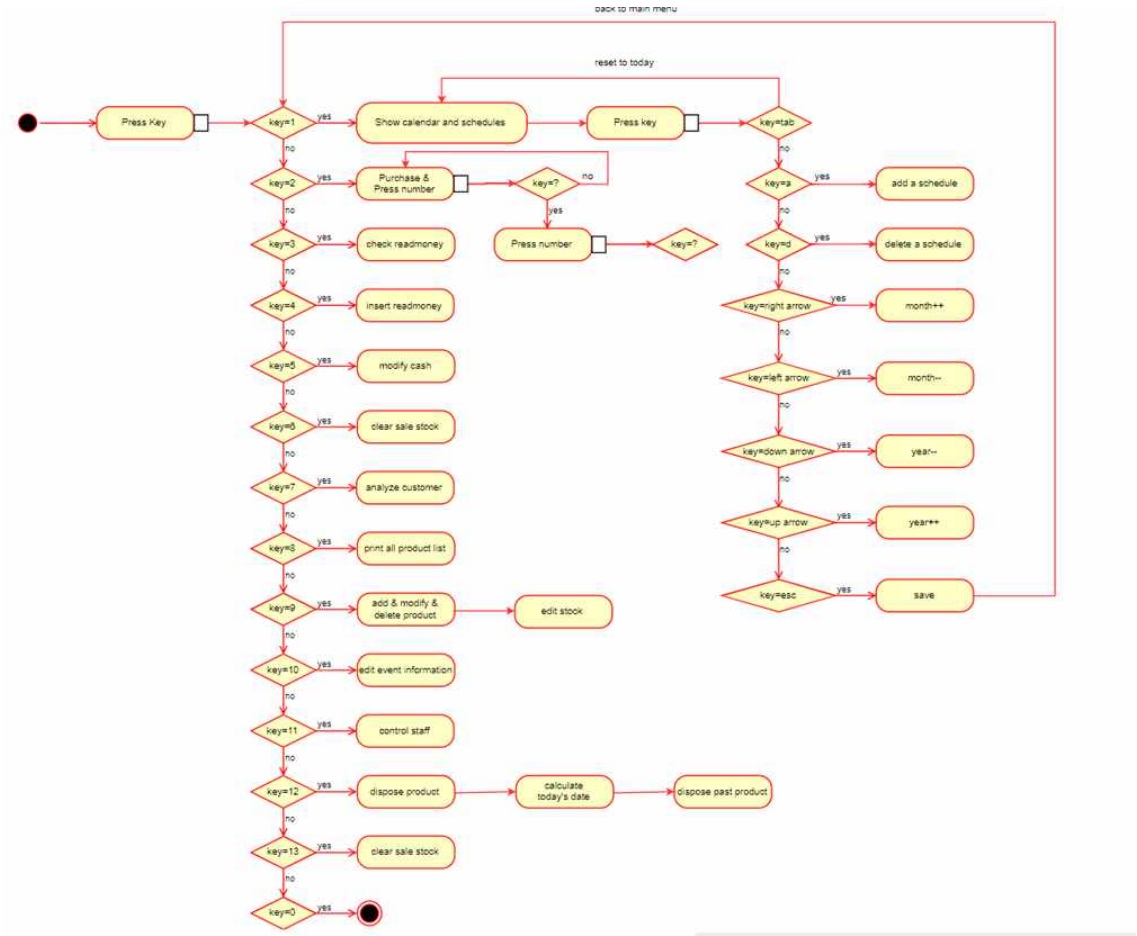

main menu(control program)
Pos
- total_money: int
- cash, card_money: int
- discount_money: int
- get_money, change: int
- product: vector<Product>
- purchase, purchase_num: vector<Product>
+ save_history(history_information, int)
+ new_customer()
+ mainMenu()
+ check_cash()
Transaction
- Tran: struct
- rep_name: string
- store_name: string
- phone_num: string
- address: string
+ add_transaction(vector<string>, vector<int>, vector<int>, vector<int>, int)
member
Staff
- Staff_schedule: struct
- num_staff: int
- new_name: char*
- negligence: int
- long_pay, holiday_pay, pay_month: int
+ get_schedule(string)
+ sort_schedule()
+ negligence()
+ pay()
Customer
- name: char*
- sex: bool
- age: int
- tel: char*
- point: int
+ Modify_name(char*), Modify_tel(char*)
+ Check_name(char*), Check_tel(char*)
+ add_point(int), sub_point(int)
+ print_information()
Product
- name: string
- price: int
- stock: vector <int>
- valid_date: vector <int>
- discount: int
- sell: int
+ setProduct(string, int, vector<int>, vector<int>, int, int)
+ addstock(int , int), substock(int)
+ get_allstock()
+ addsell(int)
+ product_discsard(int)
+ set_discount(int)


Schedule
Calendar
- Schedule: struct
- year, month, day, wday, hour, minute: int
- str_month: char
- curr_sec: time_t
- powerCal: bool
+ scheduleRead(string)
+ scheduleWrite()
+ drawDay(int)
+ keyInput()
+ drawCalendar()

(2) use case diagram



(3) activity diagram



3. 클래스

1) Calendar

```
void Calendar::reset_curr_time() {
    time(&curr_sec);
    curr_time = localtime(&curr_sec);
    year = curr_time->tm_year + 1900;
    month = curr_time->tm_mon + 1;
    day = curr_time->tm_mday;
    hour = curr_time->tm_hour;
    minute = curr_time->tm_min;
}
```

void reset_curr_time() : 캘린더에서 바뀐 연월을 현재 연월로 돌려주는 기능

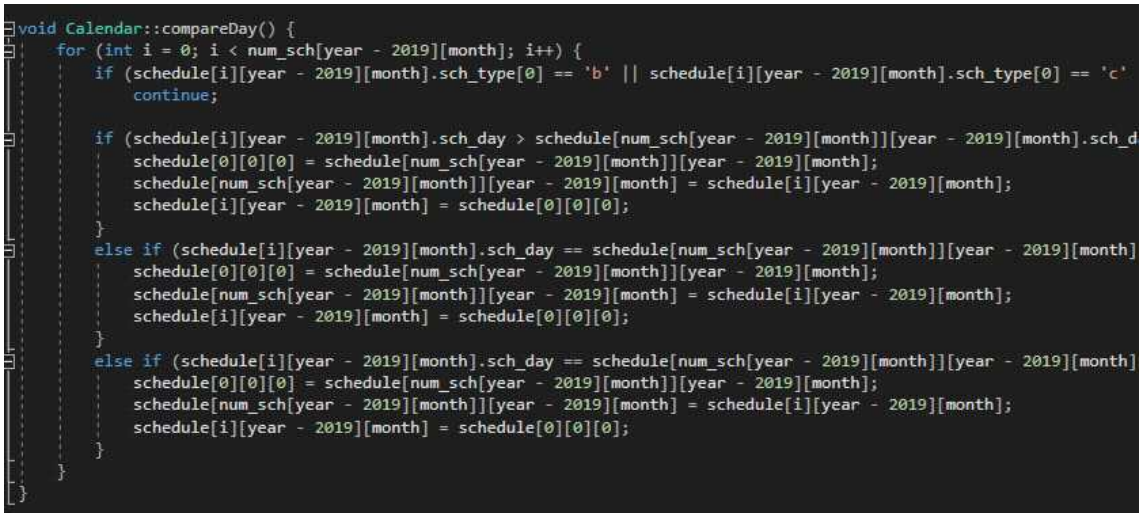
```
void Calendar::scheduleRead(string filename) {

    ifstream inFile(filename);

    while (!inFile.eof()) {
        char inputLine[100] = { 0, };
        inFile.getline(inputLine, 100);

        if (inputLine[0] == NULL) { break; }

        string r_type;
        char* r_note;

        int r_year, r_month, r_day, r_start_hour, r_start_min, r_end_hour, r_end_min;

        r_type = strtok(inputLine, "/");
        r_year = atoi(strtok(NULL, "/")) - 2019;
        r_month = atoi(strtok(NULL, "/"));
        r_day = atoi(strtok(NULL, "/"));

        schedule[num_sch[r_year][r_month]][r_year][r_month].sch_type = r_type;
        schedule[num_sch[r_year][r_month]][r_year][r_month].sch_day = r_day;
```

```
    if (r_type[0] == 'a') {
        r_start_hour = atoi(strtok(NULL, "/"));
        schedule[num_sch[r_year][r_month]][r_year][r_month].sch_start_hour = r_start_hour;


        r_start_min = atoi(strtok(NULL, "/"));
        schedule[num_sch[r_year][r_month]][r_year][r_month].sch_start_min = r_start_min;

        r_end_hour = atoi(strtok(NULL, "/"));
        schedule[num_sch[r_year][r_month]][r_year][r_month].sch_end_hour = r_end_hour;

        r_end_min = atoi(strtok(NULL, "/"));
        schedule[num_sch[r_year][r_month]][r_year][r_month].sch_end_min = r_end_min;

        r_note = strtok(NULL, "/");
        strcpy(schedule[num_sch[r_year][r_month]][r_year][r_month].sch_note, r_note);
        if(r_note == NULL)
            schedule[num_sch[r_year][r_month]][r_year][r_month].sch_note ==NULL;
    }
    else if (r_type[0] == 'e') {

    }
    else {      // b, c, d
        r_note = strtok(NULL, "/");
        strcpy(schedule[num_sch[r_year][r_month]][r_year][r_month].sch_note, r_note);
    }
    num_sch[r_year][r_month]++;

}


//sort(student, student + numStudent, compare);

inFile.close();
```

void scheduleRead(string filename) : 크리스마스나 빼빼로 데이 등 공휴일, 행사일 스케줄의 텍스트파일을 읽는 메소드

```
void Calendar::scheduleWrite() {    ////////////////////////////////////// a,e 나눠야 할수도?????????

    ofstream outFile("staff_schedule.txt");
    for (int y = 0; y < 10; y++) {
        for (int m = 1; m < 13; m++) {
            for (int i = 0; i < num_sch[y][m]; i++) {

                if (schedule[i][y][m].sch_type[0] == 'a') {
                    outFile << "a/" << y + 2019 << "/" << m << "/" << schedule[i][y][m].sch_day << "/";
                    outFile << schedule[i][y][m].sch_start_hour << "/" << schedule[i][y][m].sch_start_min << "/";
                    outFile << schedule[i][y][m].sch_end_hour << "/" << schedule[i][y][m].sch_end_min << "/";
                    outFile << schedule[i][y][m].sch_note << endl;
                }
                else {          // e일경우

                }
            }
        }
    }

    outFile.close();
}
```

void scheduleWrite() : 공휴일을 제외한 직원들 스케줄의 텍스트파일을 쓰게 해주는 메소드



```
void Calendar::compareDay() {
    for (int i = 0; i < num_sch[year - 2019][month]; i++) {
        if (schedule[i][year - 2019][month].sch_type[0] == 'b' || schedule[i][year - 2019][month].sch_type[0] == 'c'
            continue;

        if (schedule[i][year - 2019][month].sch_day > schedule[num_sch[year - 2019][month]][year - 2019][month].sch_d
            schedule[0][0][0] = schedule[num_sch[year - 2019][month]][year - 2019][month];
            schedule[num_sch[year - 2019][month]][year - 2019][month] = schedule[i][year - 2019][month];
            schedule[i][year - 2019][month] = schedule[0][0][0];
        }
        else if (schedule[i][year - 2019][month].sch_day == schedule[num_sch[year - 2019][month]][year - 2019][month]
            schedule[0][0][0] = schedule[num_sch[year - 2019][month]][year - 2019][month];
            schedule[num_sch[year - 2019][month]][year - 2019][month] = schedule[i][year - 2019][month];
            schedule[i][year - 2019][month] = schedule[0][0][0];
        }
        else if (schedule[i][year - 2019][month].sch_day == schedule[num_sch[year - 2019][month]][year - 2019][month]
            schedule[0][0][0] = schedule[num_sch[year - 2019][month]][year - 2019][month];
            schedule[num_sch[year - 2019][month]][year - 2019][month] = schedule[i][year - 2019][month];
            schedule[i][year - 2019][month] = schedule[0][0][0];
        }
    }
}
```

void compareDay() : 달력에 있는 스케줄들을 순서대로 sort해주는 메소드

```
int Calendar::D_day(int d_year, int d_month, int d_day) {
    int anni_all_day = 0, ye = 0, mo = 0, a = 0;
    int now_all_day = 0, ye2 = 0, mo2 = 0, b = 0;
    int D_day = 0;
    for (int i = 2018; i < d_year; ++i) {
        ye += 365;
        if (dayMonthLimit(2) == 29)
            ye += 366;
    }
    for (int i = 0; i < d_month - 1; ++i) {
        mo += dayMonthLimit(i);
    }
    if (dayMonthLimit(year) == 29 && d_month > 2) {
        a = 1;
    }
    anni_all_day = d_day + ye + mo + a;

    for (int i = 2018; i < year; ++i) {
        ye2 += 365;
        if (dayMonthLimit(2) == 29)
            ye2 += 366;
    }
    for (int i = 0; i < month - 1; ++i) {
        mo2 += dayMonthLimit(i);
    }
    if (dayMonthLimit(year) == 29 && month > 2) {
        b = 1;
    }
    now_all_day = day + ye2 + mo2 + b;
    D_day = (anni_all_day - now_all_day);
    return D_day;
}
```

Int D_day(int d_year, int d_month, int d_day) : 2018년부터 행사일 / 공휴일 날짜까지의 총 일수를 구하고 2018년부터 현재 일자까지의 총 일수를 빼서 D_day를 구하는 메소드

```
vector<int> Calendar::returnTime() {
    vector<int> clock = { year, month, day, hour, minute };
    return clock;
}
```

Vector<int> returnTime() : 현재의 연월일과 시간을 나타내는 메소드

2) Product

```cpp
void Product::setProduct(string name, int price, vector<int> valid_date, vector<int> stock, int discount, int sell) {
    this->setname(name);
    this->setprice(price);
    for (int i = 0; i < valid_date.size(); i++) {
        this->setvalid_date(valid_date.at(i));
        this->setstock(stock.at(i));
    }
    this->setdiscount(discount);
    this->setsell(sell);
    //this->setcategory(category);
} //배열에 등록
```

void setProduct(string name, int price, vector<int> valid_date, vector<int> stock, int discount, int sell) : 제품명,단가,할인율,판매량,유통기한과 재고를 넘겨 받아서 vector 배열에 등록하는 메소드

```cpp
void Product::addstock(int num , int date) {
    this->stock.push_back(num);
    this->valid_date.push_back(date);

}//재고 추가
```

void addstock(int num, int date) : date라는 유통기한을 가진 재고를 추가하는 메소드

```cpp
int Product::substock(int n) {
    int sum = 0;
    for (int i = 0; i < this->stock.size(); i++) {
        sum += stock[i];
    }
    if (sum < n) {
        //재고가 부족합니다 출력
    }
    else {
        while (n > 0) {
            int temp = n;
            n -= this->stock[0];
            this->stock[0] -= temp;

            if (this->stock[0] <= 0) {
                this->stock.erase(stock.begin());
                this->valid_date.erase(valid_date.begin());
            }
        }
    }
    return 1;//////////////////////////////////////////////////////////////////수정
}//재고 감소
```

Int substock(int n) : 제품이 n개 줄일때(팔렸을때), 재고가 n보다 작으면 재고부족 출력, 재고가 n보다 많으면 유통기한이 빠른순으로 재고가 빠지는 메

소드

```
void Product::addsell(int n) {
    this->sell += n;
}
```

void addsell(int n) : 제품이 n개 팔렸을 때, 제품의 판매량에 n을 더해주는 메소드

```
void Product::set_discount(int n) {
    this->discount = n;
}//물품 행사 추가
```

void set_discount(int n) : 물품의 행사 정보를 수정하는 메소드

3) Staff

```
void Staff::clear_staff() {
    for (int i = 0; i < num_staff; i++) {
        this->pay_month, holiday_pay, long_pay = 0;

        this->negligence = 0;
    }

}
```

void clear_staff() : 스태프들의 정보(월급,휴일,근무태만 등) 초기화하는 메소드

```
//현재 시간 리턴
vector<int> Staff::returnTime(){
    Calendar calendar;
    vector<int> cur_time;
    cur_time = calendar.returnTime();
    return cur_time;
}
```

vector<int> returnTime() : 현재 시간을 불러오는 메소드

```
//출근 시간 set
void Staff::set_work_start(int staff_num, int schedule_num) {
    staff2[staff_num].sch2[schedule_num].cur_work_start_time = returnTime();
}
```

void set_work_start(int staff_num, int schedule_num) : 직원의 출근시간을 설정하는 메소드

```
//퇴근시간 set
void Staff::set_work_end(int staff_num, int schedule_num) {
    staff2[staff_num].sch2[schedule_num].cur_work_end_time = returnTime();
}
```

void set_work_end(int staff_num, int schedule_num) : 직원의 퇴근시간을 설정하는 메소드

```
//근무 태만 횟수 set
void Staff::set_negligence() {
    for(int i=0;num_staff;i++)
        for (int j = 0; j < 30; j++) {
            //조기퇴근
            if ((staff2[i].sch2[j].work_end_hour - staff2[i].sch2[j].cur_work_end_time.at(4) > 0) || ((st
                staff2[i].negligence++;
            //지각
            if ((staff2[i].sch2[j].cur_work_start_time.at(4) - staff2[i].sch2[j].work_start_hour > 0) ||
                staff2[i].negligence++;
        }
}
```

void set_negligence() : 직원이 조기퇴근을 하거나 지각을 했을 시 근무태만 횟수를 증가하는 메소드

```
void Staff::calculate_work_time() { //해당스태프번호, 해당스태프의스케줄번호
    for (int staff_num = 0; staff_num < num_staff; staff_num++) {
        for (int schedule_num = 0; schedule_num < 30; schedule_num++) {
            //스태프 해당 스케줄에 일한 시간 계산
            staff2[staff_num].sch2[schedule_num].work_time_day = ((staff2[staff_num].sch2[schedule_num].w

            if (staff2[staff_num].sch2[schedule_num].work_time_day >= 8)    //8시간 이상 일하면
                staff2[staff_num].sch2[schedule_num].work_time_day = -1;    //1시간은 휴식시간

            else if (staff2[staff_num].sch2[schedule_num].work_time_day >= 4)   //4시간 이상 8시간 미만 일
                staff2[staff_num].sch2[schedule_num].work_time_day = -0.5;  //30분은 휴식시간
        }
    }
}
```

Void calculate_work_time() : 해당 직원의 일한 시간 계산하는 메소드(8시간 이상 근무시, 1시간 휴식 / 4시간 이상 8시간 미만 근무시, 30분 휴식 포함)

```
void Staff::get_schedule(string filename) {
    ifstream inFile(filename);
    while (!inFile.eof()) {
        char inputLine[100] = { 0, };
        inFile.getline(inputLine, 100);

        if (inputLine[0] == NULL) { break; }

        string r_type;

        r_type = strtok(inputLine, "/");
        staff1[num_schedule_temp].sch.work_year = atoi(strtok(NULL, "/")) - 2019;
        staff1[num_schedule_temp].sch.work_month = atoi(strtok(NULL, "/"));
        staff1[num_schedule_temp].sch.work_day = atoi(strtok(NULL, "/"));

        if (r_type[0] == 'a') {
            staff1[num_schedule_temp].sch.work_start_hour = atoi(strtok(NULL, "/"));

            staff1[num_schedule_temp].sch.work_start_min = atoi(strtok(NULL, "/"));

            staff1[num_schedule_temp].sch.work_end_hour = atoi(strtok(NULL, "/"));

            staff1[num_schedule_temp].sch.work_end_min = atoi(strtok(NULL, "/"));

            staff1[num_schedule_temp].new_name = strtok(NULL, "/");
            num_schedule_temp++;
        }
    }
    inFile.close();
}
```

void get_schedule(string filename) : 직원들의 스케줄표 텍스트파일을 읽는 메소드

```
void Staff::sort_schedule() {
    ifstream readFile;
    readFile.open("staff_list.txt");        //staff 목록

    //staff2에 직원 정렬.
    if (readFile.is_open()) {
        while (!readFile.eof()) {
            char tmp[256];
            readFile.getline(tmp, 256);
            staff2[num_staff].new_name=tmp;
            this->num_staff++;
        }
    }

    readFile.close();

    //staff2에 스케줄 정렬
    for (int i = 0; i < num_staff; i++) {
        for (int j =0; j < num_schedule_temp; j++) {
            if (!strcmp(staff2[i].new_name, staff1[j].new_name)) {
                staff2[i].sch2[num_schedule] = staff1[j].sch;
                this->num_schedule++;
            }
        }
        this->num_schedule = 0;
    }
}
```

void sort_schedule() : stafflist를 받는 메소드에서 직원들 이름을 받아서 staff2 배열에 저장한다. void get_schedule()에서 staff1배열에 스케줄을 다 넣고 위 메소드에선 staff1 배열을 이용해서 staff2 배열을 정의한다. staff1의 new_name이랑 staff2의 new_name이 같으면 sch 구조체배열에 하나씩 저장하는 방식을 따른다. 즉, sort메소드를 통해서 직원에 따라 스케줄을 정리한다

```
void Staff::add_staff() {
    char* add_staff_name;
    cout << "새로운 직원의 이름을 입력하세요 : ";
    cin >> add_staff_name;

    ofstream writeFile("staff_list.txt",ios::app);      //staff 목록
    writeFile << add_staff_name << std::endl;           //txt 파일에 직원 이름 추가
    staff2[num_staff].new_name = add_staff_name;        //staff2 배열에 직원이름 추가
    num_staff++;                                        //staff_name 증가

    writeFile.close();

}
```

void add_staff() : 새로운 직원을 추가하는 메소드

```
void Staff::delete_staff() {
    int delete_num;
    print_all();        //직원목록 전체 보여줌.

    cout << "삭제할 직원 번호를 입력하세요 : ";
    cin >> delete_num;
    this->num_staff--;

    // 예시
    // 1. 이태운 2. 이상윤 3. 이기태 4. 김태우
    // 2번을 삭제. 2번은 실제로 staff2[1]이다. 따라서, delete_num-1된 값을 삭제해야 한다.
    // num_staff는 3이다. 아직 4번은 staff[3] 저장되어 있음.
    // staff2[1]=staff[2]//이태운 이기태 이기태 김태우
    // staff2[2]=staff[3]//이태운 이기태 김태우 김태우
    for (int i = delete_num; i <= num_staff; i++) {
        staff2[i - 1] = staff2[i];
    }
}
```

void delete_staff() : 직원을 삭제하는 메소드

```
void Staff::upgrade_pay_hour() {
    for (int i = 0; i < num_staff; i++)
        if(staff2[i].negligence==0)         //negligence 횟수가 없다면
            staff2[i].pay_hour += 200;      //시급 200원 인상
}
```

void upgrade_pay_hour() : negligence가 한달동안 없으면 시급을 200원 인상하는 메소드

```
void Staff::message_negligence() {
    ofstream File("staff_negligence.txt", ios::out);        //staff negligence 목록.
    set_negligence();
    for (int i = 0; i < num_staff; i++) {
        File << staff2[i].new_name<< " 근무 태만 횟수: " << staff2[i].negligence << endl;
    }
    File.close();
}
```

void message_negligence() : 해당 직원의 근무 태만 횟수를 저장한 텍스트 파일을 읽는 메소드

```
Staff::pay() {
 ofstream file;
 file.open("staff_pay.txt", ios::app);       //staff pay 목록

for (int i = 0; i < num_staff; i++) {

    if (num_staff > 5) {                    // 직원이 5인 이상이면 연장수당, 휴일수당
        if (int j = 0; j < 20; j++) {

            //하루에 일한 시간이 8시간 초과면 연장수당
            //연장수당은 기본 시급의 0.5배
            if (staff2[i].sch2[j].work_day > 8) {
                staff2[i].long_pay += (int)((staff2[i].sch2[j].work_day - 8)*staff2[i].pay_hour*0.5);
            }

            //휴일수당은 휴일에 8시간 이하로 근무하면 기본 시급의 0.5배
            //휴일수당은 휴일에 8시간 초과하여 근무하면 기본 시급의 1배
            if (staff2[i].sch2[j].cur_work_start_time.at(3) == 0) {     //휴일에 일하였다면
                if (staff2[i].sch2[j].work_time_day > 8)
                    staff2[i].holiday_pay += (int)((staff2[i].sch2[j].work_day - 8)*staff2[i].pay_hour*1.
                else
                    staff2[i].holiday_pay += (int)((staff2[i].sch2[j].work_day - 8)*staff2[i].pay_hour*0.
            }
        }

        //월급 계산
        staff2[i].pay_month += long_pay + holiday_pay + (staff2[i].sch2[j].work_day)*staff2[i].pay_hour;

        file << staff2[i].new_name << "님의 연장수당은 " << staff2[i].long_pay << "원입니다." << endl;
        file << staff2[i].new_name << "님의 휴일수당은 " << staff2[i].holiday_pay << "원입니다." << endl;
        file << staff2[i].new_name << "님의 월급은 " << staff2[i].pay_month << "원입니다." << endl;

    }
}


file.close();
```

void pay() : 직원들에게 월급을 주는 메소드(한 주 동안 일한 시간을 초기화 시켜 급여를 매주 계산해서 쌓아가는 방식, 야간수당, 추가수당(설날, 추석 등) 계산, 휴식시간도 포함해서 근무시간을 계산하고 만약 잔액이 모자랄 경우, 알림을 주고 2주안에 지급하는 방식)

```
void Staff::print_all() {                          // 전체 직원 목록 출력
    for (int i = 0; i < num_staff; i++) {
        cout << i + 1 << ". " << staff2[i].new_name;    //직원 이름
        cout << " 시급: " << staff2[i].pay_hour << endl;    //직원 시급
    }

}
```

void print_all() : 전체 직원 목록을 출력하는 메소드

```
Staff staff1[100];  //get_schedule에서 필요한 객체.
Staff staff2[30];   //sort_schedule에서 필요한 객체.
                    //최대 직원 수: 30
                    //진짜 중요한 객체는 staff2 객체이다.
Staff::Staff() {
    get_schedule("staff_schedule.txt");
    sort_schedule();

    if (returnTime().at(2)==1) {                //한 달이 지나면(매월 1일이 되면)
        pay();
        upgrade_pay_hour();                 //시급인상을 고려한다.
        Staff::message_negligence();        //근무태만횟수를 출력한다.

        clear_staff();                      //스태프 정보 초기화.
    }

}
```

stafflist를 받는 메소드

4) POS

```
ifstream inFile(filename);

while (!inFile.eof()) {
    char inputLine[200] = { 0, };
    inFile.getline(inputLine, 200);

    if (inputLine[0] == NULL) { break; }
    string r_name;
    int r_price, r_discount, r_sell, r_category;
    //int r_valid_date;
    //int r_loacation;

    r_name = strtok(inputLine, "/");
    r_price = atoi(strtok(NULL, "/"));
    char* r_stock_index = strtok(NULL, "/");
    r_discount = atoi(strtok(NULL, "/"));
    r_sell = atoi(strtok(NULL, "/"));

    vector <int> r_valid_date;
    vector <int> r_stock;
    int r_index;
    char* r_c;

    bool c = true;
    while (1) {
        if (c) {
            r_index = atoi(strtok(r_stock_index, ".#"));
            c = false;
        }
        else {
            r_c = strtok(NULL, ".#");
            if (r_c == NULL)
                break;
            r_index = stoi(r_c);
        }

        r_valid_date.push_back(r_index);
        r_index = atoi(strtok(NULL, ".#"));
        r_stock.push_back(r_index);
    }


    Product input_p;
    input_p.setProduct(r_name, r_price, r_valid_date, r_stock, r_discount, r_sell);
    product.push_back(input_p);

}
```

void productRead(const char* filename) : 포스기 내에서 제품들이 적힌 텍스트 파일을 읽어주는 메소드

```
void POS::productWrite(const char* filename) {
    ofstream outFile(filename);

    for (int i = 0; i < product.size(); i++) {
        outFile << product.at(i).getname() << "/" << product.at(i).getprice() << "/";
        for (int j = 0; j < product.at(i).getstock().size(); j++) {
            outFile << product.at(i).getvalid_date().at(j) << "." << product.at(i).getstock().at(j);
            if (j == product.at(i).getstock().size() - 1)
                outFile << "##/";
            else
                outFile << ".";
        }
        outFile << product.at(i).getdiscount() << "/" << product.at(i).getsell() << endl;
    }
    outFile.close();

}
```

void productWrite(const char* filename) : 텍스트 파일에 제품을 써주는 메소드

```
void POS::set_total_money() {
    total_money = (cash[0] * 50000) + (cash[1] * 10000) + (cash[2] * 5000) + (cash[3] * 1000) + (cash[4]
}
```

void set_total_money() : 포스기 내에서 총 돈을 설정해주는 메소드

```
void POS::set_total_cash(int* set_cash) {
    for (int i = 0; i < 7; i++)
        cash[i] = set_cash[i];
}
```

void set_total_cash(int* set_cash) : 포스기 내의 총 돈 중 현금을 설정해주는 메소드

```
void POS::input_total_cash() {
    int set_cash[8] = { 0, };
    cout << "50000 개수 : ";
    cin >> set_cash[0];
    while (getchar() != '\n');

    cout << "10000 개수 : ";
    cin >> set_cash[1];
    while (getchar() != '\n');

    cout << "5000 개수 : ";
    cin >> set_cash[2];
    while (getchar() != '\n');

    cout << "1000원 개수 : ";
    cin >> set_cash[3];
    while (getchar() != '\n');

    cout << "500원 개수 : ";
    cin >> set_cash[4];
    while (getchar() != '\n');

    cout << "100원 개수 : ";
    cin >> set_cash[5];
    while (getchar() != '\n');

    cout << "50원 개수 : ";
    cin >> set_cash[6];
    while (getchar() != '\n');

    cout << "10원 개수 : ";
    cin >> set_cash[7];
    while (getchar() != '\n');

    set_total_cash(set_cash);
}
```

Void input_total_cash() : 포스기 내의 현금을 입력해주는 메소드

```
void POS::check_cash() {
    set_total_money();

    int check_cash[8] = { 0, };
    int check_money = 0;
    cout << "50000 개수 : ";
    cin >> check_cash[0];
    while (getchar() != '\n');

    cout << "10000 개수 : ";
    cin >> check_cash[1];
    while (getchar() != '\n');

    cout << "5000 개수 : ";
    cin >> check_cash[2];
    while (getchar() != '\n');

    cout << "1000원 개수 : ";
    cin >> check_cash[3];
    while (getchar() != '\n');

    cout << "500원 개수 : ";
    cin >> check_cash[4];
    while (getchar() != '\n');

    cout << "100원 개수 : ";
    cin >> check_cash[5];
    while (getchar() != '\n');

    cout << "50원 개수 : ";
    cin >> check_cash[6];
    while (getchar() != '\n');

    cout << "10원 개수 : ";
    cin >> check_cash[7];
    while (getchar() != '\n');

    check_money = (check_cash[0] * 50000) + (check_cash[1] * 10000) + (check_cash[2] * 5000) + (c
    if (check_money == (total_money - card_money)) {
        cout << "현금과부족 :" << 0;
    }
    else {
        cout << "현금과부족 :" << check_money - (total_money - card_money);
    }
    Sleep(1000);
```

void check_cash() : 거스름돈을 계산할 때 현금을 확인해주는 메소드

```
void POS::modify_cash() {
    int input;
    cout << "현금 입금 : 0 출금 : 1" << endl;
    cin >> input;
    while (getchar() != '\n');

    system("cls");

    if (input == 0) {
        add_total_cash();
    }
    else if (input == 1) {
        sub_total_cash();
    }
}
```

void modify_cash() : 포스기 내의 현금을 수정해주는 메소드

```
void POS::add_total_cash() {
    set_total_money();

    int input, i;
    cout << "입금할 현금 단위를 입력하시오 : ";
    cin >> input;
    while (getchar() != '\n');
    switch (input)
    {
    case 50000:
        i = 0;
        break;
    case 10000:
        i = 1;
        break;

    case 5000:
        i = 2;
        break;

    case 1000:
        i = 3;
        break;

    case 500:
        i = 4;
        break;

    case 100:
        i = 5;
        break;

    case 50:
        i = 6;
        break;

    case 10:
        i = 7;
        break;

    default:
        cout << "잘못된 단위가 입력되었습니다." << endl;
        return;
        break;
    }
```

```
    cout << "몇 개를 입금하시겠습니까?? : ";
    cin >> input;
    while (getchar() != '\n');
    if (i > 0) {
        cash[i] += input;
        set_total_money();
    }


}
```

void add_total_cash() : 포스기에서 새로운 현금을 입금해 총 현금량을 바꿔 주는 메소드

```
void POS::sub_total_cash() {
    set_total_money();

    int input, i;
    cout << "출금할 현금 단위를 입력하시오 : ";
    cin >> input;
    while (getchar() != '\n');
    switch (input)
    {
    case 50000:
        i = 0;
        break;
    case 10000:
        i = 1;
        break;

    case 5000:
        i = 2;
        break;

    case 1000:
        i = 3;
        break;

    case 500:
        i = 4;
        break;

    case 100:
        i = 5;
        break;

    case 50:
        i = 6;
        break;

    case 10:
        i = 7;
        break;

    default:
        cout << "잘못된 단위가 입력되었습니다." << endl;
        return;
        break;
    }

    cout << "몇 개를 출금하시겠습니까?? : ";
    cin >> input;
    while (getchar() != '\n');
    if (i > 0 && cash[i] >= input) {
        cash[i] -= input;
        set_total_money();
    }
    else if (i > 0 && cash[i] < input) {
        cout << "잔돈의 갯수가 모자랍니다." << endl;
    }
}
```

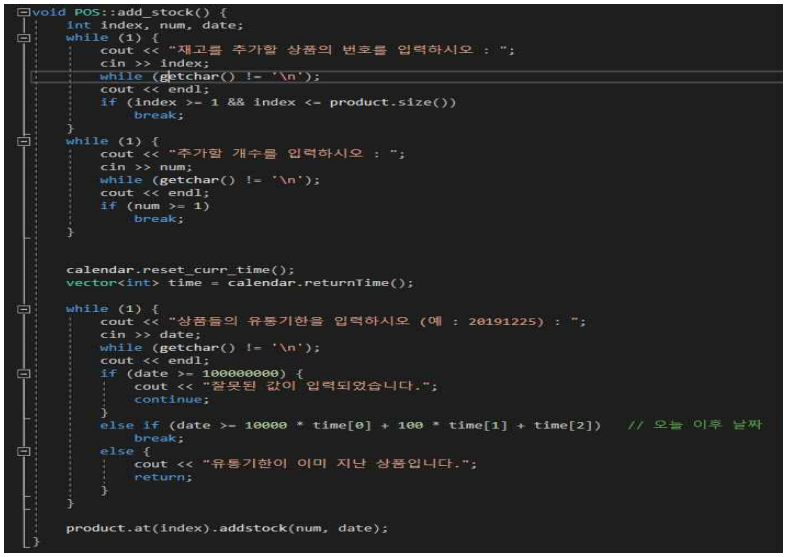
void sub_total_cash() : 포스기에서 현금을 빼내 총 현금량을 바꿔주는 메소드

```
void POS::reset_sell() {
    for (int i = 0; i < product.size(); i++)
        product.at(i).setsell(0);
}
```

Void reset_sell() : 제품의 판매량을 초기화시켜주는 메소드

```
void POS::modify_product() {
    int index;
    cout << "재고 ++ : 1" << endl;
    cout << "재고 -- : 2 - (제거는 유통기한 빠른 순으로)" << endl;
    cout << "상품 ++ : 3 - (신상품)" << endl;
    cout << "상품 -- : 4" << endl;
    cout << "메뉴를 골라주세요 : ";
    cin >> index;
    while (getchar() != '\n');

    system("cls");

    switch (index) {
    case 1:
        add_stock();
        break;
    case 2:
        sub_stock();
        break;
    case 3:
        add_product();
        break;
    case 4:
        sub_product();
        break;

    }
}
```

void modify_product() : 제품을 바꿔주는 메소드



```
void POS::add_stock() {
    int index, num, date;
    while (1) {
        cout << "재고를 추가할 상품의 번호를 입력하시오 : ";
        cin >> index;
        while (getchar() != '\n');
        cout << endl;
        if (index >= 1 && index <= product.size())
            break;
    }
    while (1) {
        cout << "추가할 개수를 입력하시오 : ";
        cin >> num;
        while (getchar() != '\n');
        cout << endl;
        if (num >= 1)
            break;
    }


    calendar.reset_curr_time();
    vector<int> time = calendar.returnTime();

    while (1) {
        cout << "상품들의 유통기한을 입력하시오 (예 : 20191225) : ";
        cin >> date;
        while (getchar() != '\n');
        cout << endl;
        if (date >= 100000000) {
            cout << "잘못된 값이 입력되었습니다.";
            continue;
        }
        else if (date >= 10000 * time[0] + 100 * time[1] + time[2])   // 오늘 이후 날짜
            break;
        else {
            cout << "유통기한이 이미 지난 상품입니다.";
            return;
        }
    }

    product.at(index).addstock(num, date);
}
```

Void add_stock() : 제품의 재고를 추가해주는 메소드

```
void POS::sub_stock() {
    int index, num, date;
    while (1) {
        cout << "재고를 제거할 상품의 번호를 입력하시오 : ";
        cin >> index;
        while (getchar() != '\n');
        cout << endl;
        if (index >= 1 && index <= product.size())
            break;
    }

    while (1) {
        cout << "제거할 개수를 입력하시오 : ";
        cin >> num;
        while (getchar() != '\n');
        cout << endl;
        if (num >= 1 && num <= product.at(index).get_allstock())
            break;
    }
    product.at(index).substock(num);

    cout << "제거 완료";
    Sleep(1000);
}
```

Void sub_stock() : 제품의 재고를 유통기한이 빠른순으로 빼는 메소드

```
void POS::add_product() {
    string i_name;
    int i_price, i_stock, i_valid_date;
    int i_sell = 0;
    int i_discount = 0;

    cout << "추가할 상품의 이름을 입력하시오 : ";
    cin >> i_name;
    while (getchar() != '\n');
    //strcpy(schedule[num_sch[year - 2019][month]][year - 2019][month].sch_note, i_name.c_str());

    while (1) {
        cout << "추가할 상품의 가격을 입력하시오 : ";
        cin >> i_price;
        while (getchar() != '\n');
        cout << endl;
        if (i_price >= 10 && i_price % 10 == 0)
            break;
        else {
            cout << "가격은 10의 배수로 설정해 주세요." << endl;
        }
    }

    while (1) {
        cout << "추가할 개수를 입력하시오 : ";
        cin >> i_stock;
        while (getchar() != '\n');
        cout << endl;
        if (i_stock >= 1)
            break;
    }


    calendar.reset_curr_time();
    vector<int> time = calendar.returnTime();
```

```
    while (1) {
        cout << "상품들의 유통기한을 입력하시오 (예 : 20191225) : ";
        cin >> i_valid_date;
        while (getchar() != '\n');
        cout << endl;
        if (i_valid_date >= 100000000) {
            cout << "잘못된 값이 입력되었습니다.";
            continue;
        }
        else if (i_valid_date >= 10000 * time[0] + 100 * time[1] + time[2])   // 오늘 이후 날짜
            break;
        else {
            cout << "유통기한이 이미 지난 상품입니다.";
            return;
        }
    }

    while (1) {
        cout << "추가할 상품의 행사정보를 입력하시오 : ";
        cin >> i_discount;
        while (getchar() != '\n');
        cout << endl;
        if (i_discount >= 0)
            break;
    }

    vector <int> dumy;
    Product input_p;
    input_p.setProduct(i_name, i_price, dumy, dumy, i_discount, i_sell);
    input_p.addstock(i_stock, i_valid_date);

    product.push_back(input_p);
```

void add_product() : 새로운 제품을 추가해주는 메소드

```
void POS::sub_product() {
    int index, num, date;
    while (1) {
        cout << "제거할 상품의 번호를 입력하시오 : ";
        cin >> index;
        while (getchar() != '\n');
        cout << endl;
        if (index >= 1 && index <= product.size())
            break;
    }
    product.erase(product.begin() + index - 1);
}
```

void sub_product() : 제품을 삭제해주는 메소드

```
void POS::input_buy_product() {

    char in[100];
    int input;

    sum_purchase = 0;
    purchase.clear();
    purchase_num.clear();

    while (1) {                     // 구매할 상품들 바코드(숫자)로 입력
        cin >> in;
        while (getchar() != '\n');
        if (in[0] == '?')            //아무것도없이 엔터만 쳤을때
            break;


        input = atoi(in);
        purchase.push_back(input);
        purchase_num.push_back(1);

        cout << "행사 - " << discount_string(product.at(input).getdiscount()) << endl;


        for (int i = 0; i < purchase.size() - 1; i++) {            //같은 제품일 경우 수량만 up
            if (input == purchase[i]) {
                purchase_num[i]++;
                purchase.pop_back();
                purchase_num.pop_back();
            }

        }
    }
```

```
if (purchase.size() == 0) {
    cout << "구매 상품이 없습니다!" << endl;
    return;
}
for (int i = 0; i < purchase.size(); i++) {
    if (this->product[purchase[i]].get_allstock() < purchase_num[i]) {
        cout << "재고가 부족합니다 (" << purchase[i] << ")" << endl;
        purchase.erase(purchase.begin() + i);
    }
}

for (int i = 0; i < purchase.size(); i++) {
    if(product.at(i).getdiscount() < 10 && product.at(i).getdiscount() > 0)
        sum_purchase += (product[purchase[i]].getprice() * purchase_num[i] / 10 * 1);
    else
        sum_purchase += (product[purchase[i]].getprice() * purchase_num[i]);
}

while (1) {
    cout << "현금 결제 - 0 , 카드 결제 - 1 : " << endl;
    cin >> input;
    while (getchar() != '\n');
    if (input == 0 || input == 1)
        break;
}
int card_cash = input;

if (card_cash) {                // 카드 결제
    this->card_money += this->sum_purchase;      //카드 머니 증가

    for (int i = 0; i < purchase.size(); i++) {
        product[purchase[i]].addsell(purchase_num[i]);
        product[purchase[i]].substock(purchase_num[i]);     //재고 감소
        if (product.at(purchase[i]).getdiscount() == 11) {
            product[purchase[i]].addsell(purchase_num[i]);
            product[purchase[i]].substock(purchase_num[i]);
        }
        else if (product.at(purchase[i]).getdiscount() == 23) {
            product[purchase[i]].addsell(purchase_num[i] / 2);
            product[purchase[i]].substock(purchase_num[i] / 2);
        }
    }
```

```
else {                  // 현금 결제
    while (1) {
        cout << "받은 금액 입력 : " << endl;
        cin >> this->get_money;
        while (getchar() != '\n');
        //qt에서 추가 현금 얼마짜리로 몇장 받는지
        if (this->get_money >= sum_purchase) {

            break;
        }
        cout << "금액이 부족합니다" << endl;
    }
    this->change = this->get_money - sum_purchase;            //거스름돈
    this->calculate_cash_buy(this->change);                   //현금 낼 거스름돈 계산?
    for (int i = 0; i < purchase.size(); i++) {
        product[purchase[i]].addsell(purchase_num[i]);
        product[purchase[i]].substock(purchase_num[i]);         //재고 감소
        if (product.at(purchase[i]).getdiscount() == 11) {
            product[purchase[i]].addsell(purchase_num[i]);
            product[purchase[i]].substock(purchase_num[i]);
        }
        else if (product.at(purchase[i]).getdiscount() == 21) {
            product[purchase[i]].addsell(purchase_num[i] / 2);
            product[purchase[i]].substock(purchase_num[i] / 2);
        }
    }
}
Sleep(5000);
//영수증 - purchase 이용해서!!
```

void input_buy_product() : 포스기에서 계산하는 과정, 현금/카드 결제를 결정하고 현금일 때 거스름돈까지 계산해주는 메소드

```
void POS::calculate_cash_buy(int q) {

    set_total_money();

    // (고객지불금액) — (제품가격 * 제품개수) 가정
        // (고객지불금액) — (제품가격 * 제품개수) 해서 거스름돈 계산
    cout << "거스름돈 : " << q << endl;
    if (cash[0] < q / 50000) {
        cout << "50000원 : " << cash[0];
        cout << "\t50000원이 부족합니다." << endl;
        q -= (cash[0] * 50000);
        cash[0] = 0;

    }
    else {
        cout << "50000원 :    " << q / 50000 << endl;
        cash[0] = cash[0] - (q / 50000);
        q = q % 50000;
    }


    if (cash[1] < q / 10000) {
        cout << "10000원 :   " << cash[1];
        cout << "\t10000원이 부족합니다." << endl;
        q -= (cash[1] * 10000);
        cash[1] = 0;
    }
    else {
        cout << "10000원 :    " << q / 10000 << endl;
        cash[1] = cash[1] - (q / 10000);
        q = q % 10000;
    }


    if (cash[2] < q / 5000) {
        cout << "5000원 :   " << cash[2];
        cout << "\t5000원이 부족합니다." << endl;
        q -= (cash[2] * 5000);
        cash[2] = 0;
    }
    else {
        cout << "5000원 :    " << q / 5000 << endl;
        cash[2] = cash[2] - (q / 5000);
        q = q % 5000;
    }
```

```
        cash[4] = 0;
    }
    else {
        cout << "500원 :  " << q / 500 << endl;
        cash[4] = cash[4] - (q / 500);
        q = q % 500;
    }


    if (cash[5] < q / 100) {
        cout << "100원 :  " << cash[5];
        cout << "\t100원이 부족합니다." << endl;
        q -= (cash[5] * 100);
        cash[5] = 0;
    }
    else {
        cout << "100원 :  " << q / 100 << endl;
        cash[5] = cash[5] - (q / 100);
        q = q % 100;
    }


    if (cash[6] < q / 50) {
        cout << "50원 :  " << cash[6];
        cout << "\t50원이 부족합니다." << endl;
        q -= (cash[6] * 50);
        cash[6] = 0;
    }
    else {
        cout << "50원 :  " << q / 50 << endl;
        cash[6] = cash[6] - (q / 50);
        q = q % 50;
    }


    if (cash[7] < q / 10) {
        cout << "10원 :  " << cash[7];
        cout << "\t10원이 부족합니다." << endl;
        q -= (cash[7] * 10);
        cash[7] = 0;
    }
    else {
        cout << "10원 :  " << q / 10 << endl;
        cash[7] = cash[7] - (q / 10);
    }

    if (total_money - card_money < q) {
```

void calculate_cash_buy(int q) : 현금 계산 시 50000원 / 10000원 / 5000원 / 1000원 / 500원 / 100원 / 50원 / 10원 별로 거스름돈을 계산해주는 메소드

```
void POS::modify_discount() {

    int i_index;
    int i_discount = 0;

    cout << "0 : 행사 없음 | 1~9 : 10% ~ 90% 할인 | 11 : 1+1 행사 | 21 : 2+1 행사" << endl;
    cout << "행사 정보를 수정할 상품의 번호를 입력하시오 : ";
    cin >> i_index;
    while (getchar() != '\n');

    while (1) {
        cout << "수정할 행사 정보를 입력하시오 : ";
        cin >> i_discount;
        while (getchar() != '\n');
        cout << endl;
        if (i_discount >= 0 && i_discount < 10)
            break;
        else if (i_discount == 11 || i_discount == 21)
            break;
        else {
            cout << "잘못된 값이 입력되었습니다." << endl;
        }
    }
    product.at(i_index).setdiscount(i_discount);
}
```

Void modify_discount() : 제품의 행사 정보를 수정해주는 메소드

```
string POS::discount_string(int dis) {
    switch (dis) {
    case 0:
        return "행사 없음";
    case 1:
        return "10% 할인";
    case 2:
        return "20% 할인";
    case 3:
        return "30% 할인";
    case 4:
        return "40% 할인";
    case 5:
        return "50% 할인";
    case 6:
        return "60% 할인";
    case 7:
        return "70% 할인";
    case 8:
        return "80% 할인";
    case 9:
        return "90% 할인";
    case 11:
        return "1+1 행사";
    case 21:
        return "2+1 행사";

    }
}
```

void discount_string(int dis) : 10% 할인이나 1+1 할인 등 제품의 행사 정보를 정해주는 메소드

```
void POS::trash_product_clear() {
    int index, date;

    calendar.reset_curr_time();
    vector<int> time = calendar.returnTime();


    cout << "***유통기한 지난 제품 목록***" << endl;
    int num = 0;
    for (int i = 0; i < product.size(); i++) {
        for (int j = 0; j < product.at(i).getstock().size(); j++) {

            if (product.at(i).getvalid_date().at(j) < 10000 * time[0] + 100 * time[1] + time[2])
                continue;
            else {
                cout << i + 1 << ". " << product.at(i).getname() << endl;
                cout << product.at(i).getstock().at(j) << "개 - ";
                cout << product.at(i).getvalid_date().at(j) << "까지" << endl;
                num++;
            }



        }
        product.at(i).substock(num);
    }


    Sleep(5000);
    cout << endl << "제거 완료";
    Sleep(2000);
    productWrite("Product.txt");
}
```

Void trash_product_clear() : 유통기한이 지난 제품들을 폐기해주는 메소드

5) Transaction

```
void Transaction::add_transaction(vector<string> name, vector<int> price, vector<int> count, vector<int>
    Tran input_t;

    input_t.number = transact.size() + 1;
    for (int i = 0; i < name.size(); i++) {
        input_t.name.push_back(name.at(i));
        input_t.price.push_back(price.at(i));
        input_t.count.push_back(count.at(i));
        input_t.discount.push_back(discount.at(i));
    }
    input_t.sum_price = sum_price;


    transact.push_back(input_t);

}
```

Void add_transaction(vector<string> name, vector<int> price, vector<int> count, vector<int> discount, int sum_price) : 거래내역을 추가하는 메소드

```
void Transaction::transactionRead(string filename) {

    ifstream inFile(filename);

    while (!inFile.eof()) {

        Tran input_t;

        char inputLine[200] = { 0, };
        inFile.getline(inputLine, 200);

        if (inputLine[0] == NULL) { break; }
        vector<string> r_name;
        vector<int> r_price;
        vector<int> r_count;
        vector<int> r_discount;
        int r_number, r_sum_price;

        string r_str;

        input_t.number = atoi(strtok(inputLine, "/"));
        char* r_product_index = strtok(NULL, "/");
        input_t.sum_price = atoi(strtok(NULL, "/"));


        int r_index;
        char* r_c;
```

```
    while (1) {
        if (c) {
            r_c = strtok(r_product_index, ".#");
            c = false;
        }
        else {
            r_c = strtok(NULL, ".#");
            if (r_c == NULL)
                break;
            r_index = atoi(r_c);
        }

        input_t.name.push_back(r_c);

        r_index = atoi(strtok(NULL, ".#"));
        input_t.price.push_back(r_index);

        r_index = atoi(strtok(NULL, ".#"));
        input_t.count.push_back(r_index);

        r_index = atoi(strtok(NULL, ".#"));
        input_t.discount.push_back(r_index);
    }


    transact.push_back(input_t);

}
```

Void transactionRead(string filename) : 거래내역이 적힌 텍스트 파일을 읽어오는 메소드

```
void Transaction::transactionWrite(string filename) {
    ofstream outFile(filename);

    for (int i = 0; i < transact.size(); i++) {
        outFile << transact.at(i).number << "/";
        for (int j = 0; j < transact.at(i).name.size(); j++) {
            outFile << transact.at(i).name.at(j) << "." << transact.at(i).price.at(j) << ".";
            outFile << transact.at(i).count.at(j) << "." << transact.at(i).discount.at(j);
            if (j == transact.at(i).name.size() - 1)
                outFile << "##/";
            else
                outFile << ".";
        }
        outFile << transact.at(i).sum_price << endl;
    }
    outFile.close();

}
```

Void transactionWrite(string filename) : 거래내역 텍스트 파일을 쓰는 메소드

```cpp
void Transaction::print_receipt(int index) {        // 영수증 출력

    cout << " ************************영수증****************************" << endl;
    cout << "대표자 이름 : " << rep_name << endl;
    cout << "가게 이름 : " << store_name << endl;
    cout << "전화번호 : " << phone_num << endl;
    cout << "가게 주소 : " << address << endl;
    cout << "와이파이 비밀번호 : emart3474 " << endl;
    cout << "***\t 품명 \t 단가 \t 수량 \t 행사 \t\t 금액 ***" << endl;
    for (int i = 0; i < transact.at(index).name.size(); i++) {
        cout << i + 1 << ".\t" << transact.at(index).name.at(i) << "\t" << transact.at(index).price.at(i
        cout << discount_string(transact.at(index).discount.at(i)) << "\t" << transact.at(index).price.a

    }
    cout << "총 금액 : " << transact.at(index).sum_price << endl;
    cout << " ********************************************************" << endl;
    ////////////////////////////////////////////////////////////////////////////////////////////////////




}
```

Void print_receipt(int intdex) : 영수증을 출력해주는 메소드

6) Customer

```
void Customer::Modify_name(char* _name)
    this->name = _name;
}
```

Void Modify_name(char* _name) : 고객의 이름을 설정하는 메소드

```
void Customer::Modify_tel(char* _tel) {
    this->tel = _tel;
}
```

Void Modify_tel(char* _tel) : 고객의 전화번호를 설정하는 메소드

```
bool Customer::Check_name(char* _name) {
    if (name == _name) return true;
    else return false;
}
```

Void check_name(char* _name) : 고객의 이름이 저장된 이름과 맞는지 확인해주는 메소드

```
bool Customer::Check_tel(char* _tel) {
    int count = 0;
    for (int index = 0; index < 4; index++)
        if (tel[index] == _tel[index]) count++;
    if (count == 4) return true;
    else return false;
}
```

Void check_tel(char* _tel) : 고객의 전화번호가 저장된 전화번호와 맞는지 확인해주는 메소드

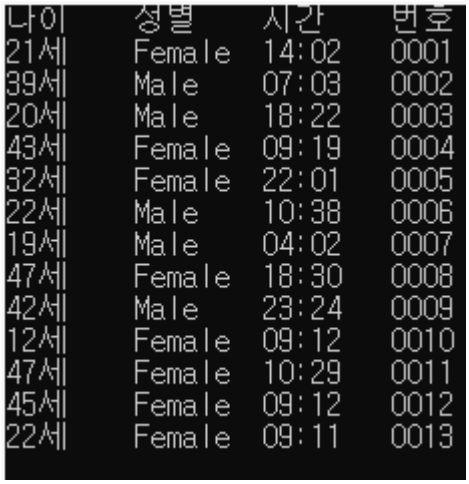
```
void Customer::add_point(int num) {
    this->point += num;
}
void Customer::sub_point(int num) {
    if (this->point < num) PRINT_POINT_ERROR
    else this->point -= num;
}
void Customer::add_age() {
    this->age++;
}
```

Void add_point(int num) : 포인트를 더해주는 메소드

Void sub_point(int num) : 포인트를 차감하는 메소드

4. 특징

1) 고객의 나이, 성별 등의 분석을 통해 고객 관리 가능



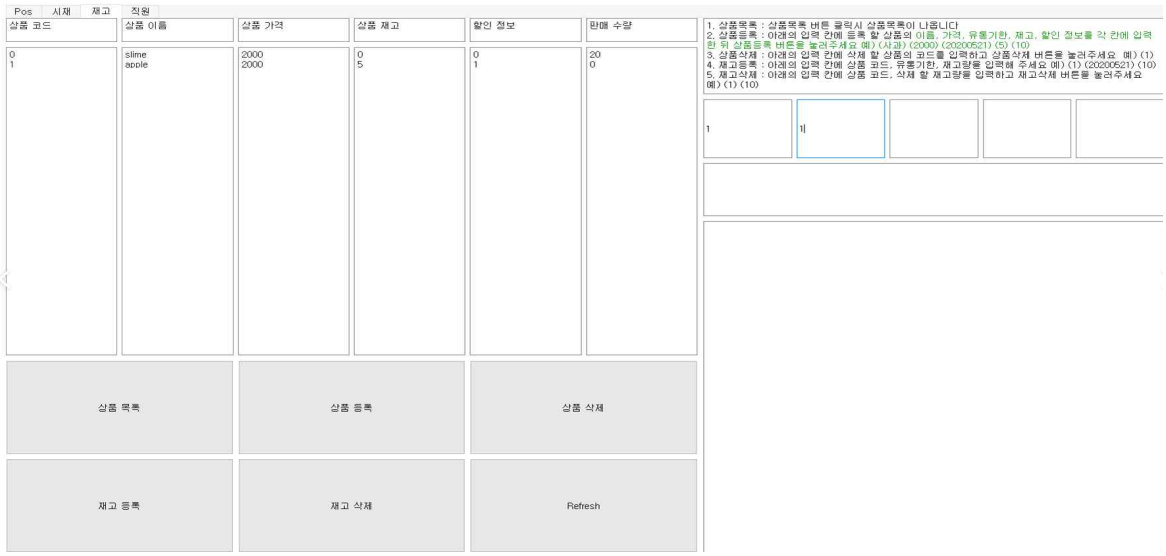
나이	성별	시간	번호
21세	Female	14:02	0001
39세	Male	07:03	0002
20세	Male	18:22	0003
43세	Female	09:19	0004
32세	Female	22:01	0005
22세	Male	10:38	0006
19세	Male	04:02	0007
47세	Female	18:30	0008
42세	Male	23:24	0009
12세	Female	09:12	0010
47세	Female	10:29	0011
45세	Female	09:12	0012
22세	Female	09:11	0013

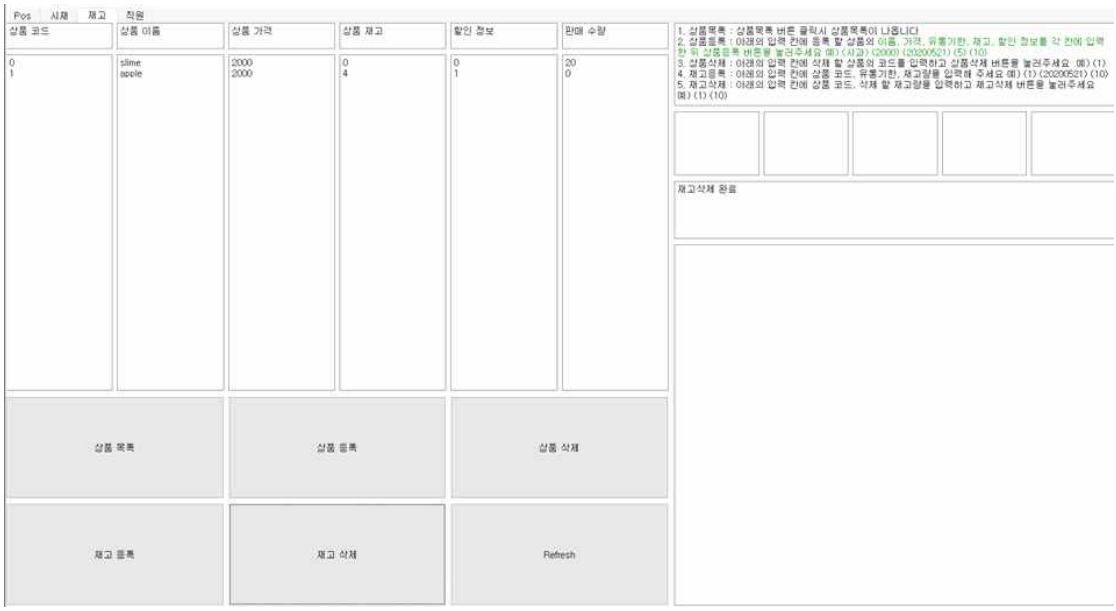
위 사진처럼 고객의 age / sex 등을 통해 해당 고객을 관리할수 있다.

2) 포인트 적립/사용 가능

계산할 때 총금액의 1% 만큼 포인트를 저장해서 해당 고객 객체의 포인트를 올려준다.

3) 재품 재고 관리 및 폐기 가능



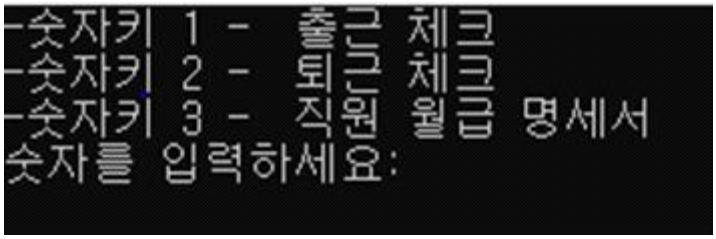


첫 번째 사진처럼 slime 제품의 재고는 0개이고, apple 제품의 재고는 5개인 것처럼 제품들의 재고를 관리할 수 있다. 또 두 번째 사진을 보면 재고삭제 버튼을 누르면 재고삭제 완료라는 알림이 뜨고 apple의 재고가 5 -> 4가 된 걸 볼 수 있다. 이처럼 재품 재고 관리 및 폐기가 가능하게 구현했다.

4) 10% 할인 등 행사제품 배치

위 사진을 보면 할인 정보 목록이 있는 걸 볼 수 있다. 할인 정보에 0~9까지 숫자가 적혀있는데 1은 10% 할인, 9는 90%할인 이라는 뜻이다. 할인 정보가 저장된 제품은 포스기에서 계산될 때 자동으로 할인이 된다.

5) 직원들 출근/퇴근 시간 관리를 통해 지각,근무태만을 점검하고, 이를 기반으로 임금인상을 결정



위 사진처럼 직원들이 출/퇴근 할때 포스기에 찍음으로써 해당 직원이 지각을 했는지 알 수 있게 구현하였다. 또 직원들이 지각/근무태만 등을 점검하여 임금인상을 결정할 수 있게 구현했다.

6) 직원들 업무시간 계산을 통해 임금 지급(세금/연장수당,휴일수당 포함)

해당 직원의 업무시간을 계산을 통해 임금을 지급한다. 이때 임금에는 세금/연장수당, 휴일 수당을 포함한다.

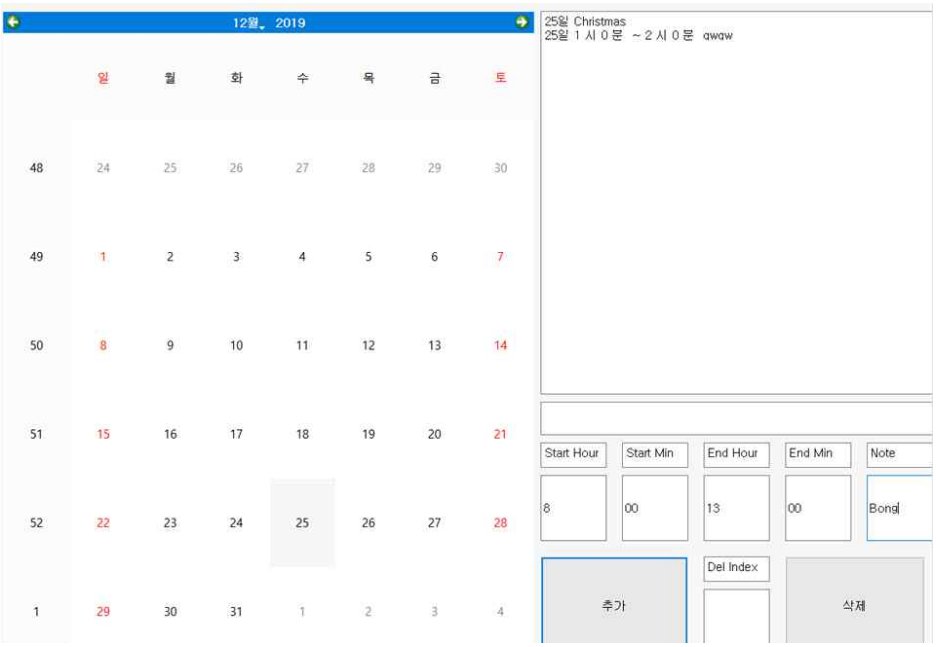
7) 직원추가, 삭제 가능

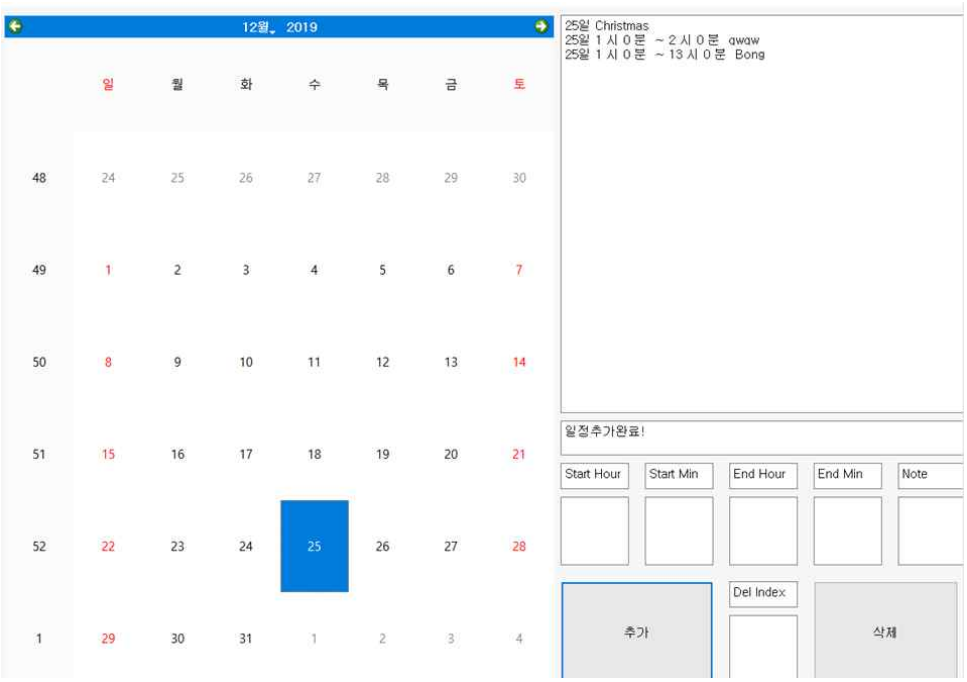
새로운 직원을 추가하고, 삭제할 수 있다.

8) 직원들 명세서 관리

전체 직원의 월급 명세서 텍스트파일(staff_pay.txt)을 볼 수 있다.
단, 한 달이 지나야 볼 수 있다.
(캡처 없음)

9) 각 행사일 / 공휴일 / 직원들 스케줄 달력에 관리 가능





첫 번 째 사진처럼 크리스마스, 신년, 빼빼로데이 같은 각 공휴일 / 행사일들을 포스기 달력에서 볼 수 있으며 두 번째 사진처럼 Start Hour / Start Min

/ End Hour / End Min / Note를 입력하면 일정추가완료! 알림이 뜨며 직원들 근무 스케줄 관리가 가능하다.

10) 포스기 내에서 거래내역을 저장해 각 거래내역의 영수증 출력 가능

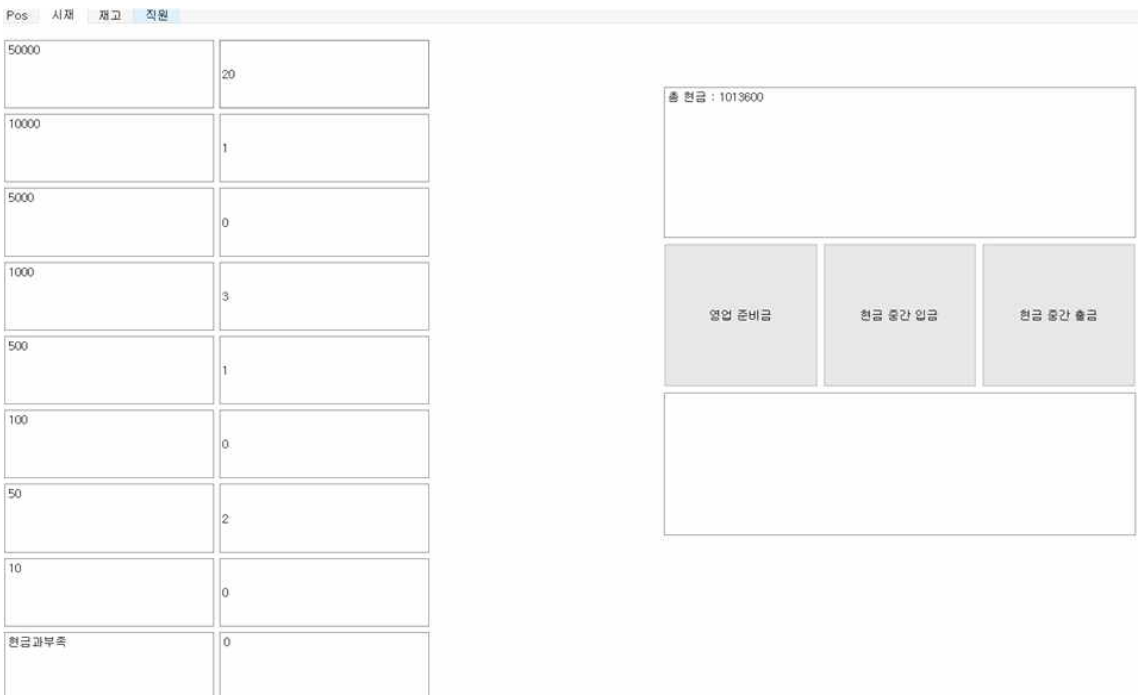
포스기 내에서 거래내역들을 보고 해당 거래내역의 영수증을 출력할 수 있다.

11) 포스기 내에서 현금/카드 결제를 나눌수 잇고 거스름돈 계산 가능

합계 수량	1
합계 금액	2000
할인 금액	200
받을 금액	1800
받은 금액	10000
거스름돈	8200

제품을 계산할 때 첫 번째 사진처럼 카드/현금을 클릭해 결제수단을 정할 수 있고 두 번째 사진처럼 받을 금액이 1800일 때, 10000을 지불했으면 8200이 거스름돈으로 나오는 걸 볼 수 있다.

12) 시재(현금) 관리 가능



위 사진처럼 현재 포스기에 있는 현금 즉 시재를 관리 할 수 있다. 위 사진은 현재 포스기에 1013600원이 있는 상태이며 50000원 20개, 10000원 1개, 5000원 0개, 1000원 3개, 500원 1개, 100원 0개, 50원 2개, 10원 0개처럼 각 현금 단위 별로 자세히 볼 수 있다.

13) 제품의 판매량 관리 가능

상품 코드	상품 이름	상품 가격	상품 재고	할인 정보	판매 수량
0	slime	2000	5	0	25
1	apple	2000	1	1	3

위 사진처럼 slime의 판매 수량은 25개, apple의 판매 수량은 3개를 볼 수 있는 것처럼 모든 제품의 판매량을 관리할 수 있다.

14) 제품 추가 삭제 가능

1. 상품목록 : 상품목록 버튼 클릭시 상품목록이 나옵니다
2. 상품등록 : 아래의 입력 칸에 등록 할 상품의 이름, 가격, 유통기한, 재고, 할인 정보를 각 칸에 입력한 뒤 상품등록 버튼을 눌러주세요 예) (사과) (2000) (20200521) (5) (10)
3. 상품삭제 : 아래의 입력 칸에 삭제 할 상품의 코드를 입력하고 상품삭제 버튼을 눌러주세요 예) (1)
4. 재고등록 : 아레의 입력 칸에 상품 코드, 유통기한, 재고량을 입력해 주세요 예) (1) (20200521) (10)
5. 재고삭제 : 아래의 입력 칸에 상품 코드, 삭제 할 재고량을 입력하고 재고삭제 버튼을 눌러주세요 예) (1) (10)

americano	1800	20200213	50	0

상품 코드	상품 이름	상품 가격	상품 재고	할인 정보	판매 수량
0	slime	2000	5	0	25
1	apple	2000	1	1	3
2	americano	1800	50	0	0

첫 번째 사진처럼 안내문을 따라 입력하면 제품 추가/삭제가 가능하다.
예를 들어 두 번째 사진처럼 americano라는 제품을 1800원에 유통기한 20200213까지 재고 50개 할인 없이 추가를 하면 세 번째 사진처럼 americano 제품이 추가되는 것을 볼 수 있다.

5. 보완할 점

1. 캘린더에 스케줄을 추가할 때 <Michael Jordan> 같은 띄어쓰기 입력이 불가능해 외국인 직원 추가 불가

2. 알바 대타 기능을 추가하면 조금 더 완벽한 포스기 가능

3. 포스기 내에서 각 제품의 판매량을 분석해 월별 BEST 5 / WORST 5 기능 추가

4. 행사나 공휴일이 가까워지면 그와 관련된 제품 재고 추가를 권유하는 알림 기능 추가 (ex — “곧 빼빼로데이입니다. 빼빼로의 재고를 추가해주세요”)

5. 포스기 내에서 문서 관리(ex 월급 명세서) 기능 추가

6. 제품별 카테고리(과자/과일/음료) 분류 기능 추가

7. 매대 재고랑 창고 재고 분리 기능 추가

8. 일부 비주얼스튜디오에선 구현이 되지만 qt에서는 구현이 안되는 기능들 수정