UnitTest

Electronic Door Lock System

7 Team

200810773 차소익, 201011364 정광용

Reference

- Unit Test Case 문서 및 기타 문서 참조
- http://dslab.konkuk.ac.kr/Class/2013/13SE/ClassB/13SE_B.htm

Assertion()

- 상태변화와 리턴값에 대해 기대값과 비교하는 함수를 모두 제작

```
void Assertion(char*, int, int);
void Assertion(char*, int, int, int, int);
void Assertion(char*, int, bool, int, bool);
void Assertion(char*, int, int, int, int, int, int);
void Assertion(char*, int, int, bool, int*, int, int, bool, int*);
void Assertion(char*, int*, bool, int*, bool);
void Assertion(char*, int*, bool, bool, int*, bool, bool);
void Assertion(char*, int, int, int*, int, int, int*);
```

Assertion()

```
void Assertion(char* id, int num1, int expected1)
{
    if ( num1 == expected1 )
    {
        cout << id << " : Pass" << endl;
    }
    else
    {
        cout << id << " : Fail" << endl;
    }
}
void Assertion(char* id, int num1, int num2, int num3, int expected1, int expected2, int expected3)
{
    if ( num1 == expected1 && num2 == expected2 && num2 == expected3 )
    {
        cout << id << " : Pass" << endl;
    }
    else
    {
        cout << id << " : Fail" << endl;
    }
}
```

LockUnlockControl()

```
void LockUnlockControl()
{
    int Input = DetermineLockUnlock();

    if ( Input == 0 )
    {
        UnlockInterface();
    }
    else if ( Input == 1 )
    {
        LockInterface();
    }
}
```

```
void LockUnlockControl(int& Triggered, int Input_I)
{
    int Input = Input_I;

    if ( Input == 0 )
    {
        Triggered = 0;
    }
    else if ( Input == 1 )
    {
        Triggered = 1;
    }
    else
    {
        Triggered = -1;
    }
}
```

AlertControl()

```
void AlertControl()
{
    int Input_1 = DetermineAlert();
    int Input_2 = PasswordTimeover();

    if ( Input_1 == 1 )
    {
        Alert1Interface();
    }
    else if ( Input_1 == 2 )
    {
        Alert2Interface();
    }
    else if ( Input_1 == 3 || Input_2 == 3 )
    {
        Alert3Interface();
    }
}
```

```
void AlertControl(int& Triggered, int Input_I)
{
    int Input = Input_I;

    if ( Input == 1 )
    {
        Triggered = 1;
    }
    else if ( Input == 2 )
    {
        Triggered = 2;
    }
    else if ( Input == 3)
    {
        Triggered = 3;
    }
    else
    {
        Triggered = -1;
    }
}
```

BacklightControl()

```
void BacklightControl()
{
    int Input_1 = DetermineBacklight();
    int Input_2 = BacklightOnPW();
    int Input_3 = BacklightOffPW();

    if ( Input_1 == 0 || Input_3 == 1 )
    {
        LightOffInterface();

        for (int i = 0; i < 5; i++)
        {
            InputData[i] = -1;
        }
    }
    else if ( Input_1 == 1 || Input_2 == 1)
    {
        LightOnInterface();

        for (int i = 0; i < 5; i++)
        {
            InputData[i] = -1;
        }
    }
    sq = -1;
}
```

```
void BacklightControl(int& Triggered, int& sq_S, int* InputData_S, int Input_I)
{
    int Input = Input_I;

    if ( Input == 0 )
    {
        Triggered = 0;
        for (int i = 0; i < 5; i++)
        {
            InputData_S[i] = -1;
        }
        sq_S = -1;
    }
    else if ( Input == 1 )
    {
        Triggered = 1;
        for (int i = 0; i < 5; i++)
        {
            InputData_S[i] = -1;
        }
        sq_S = -1;
    }
    else
    {
        Triggered = -1;
    }
}
```

DetermineLockUnlock()

```
int DetermineLockUnlock()
{
    ClosedSensorOpened();
    ClosedSensorClosed();
    int Lock_1 = ClosedSensorLock();
    int Lock_2 = ManualLockReceiver();
    int Unlock_1 = KeyReceiver();
    int Unlock_2 = NumberCheck(true);

    if ( Lock_1 == 1 || Lock_2 == 1)
    {
        return 1;
    }
    else if ( Unlock_1 == 1 || Unlock_2 == 1 )
    {
        return 0;
    }
    else
    {
        return -1;
    }
}
```

```
void DetermineLockUnlock(int& Returned, int Lock_I, int Unlock_I)
{
    int Lock = Lock_I;
    int Unlock = Unlock_I;

    if ( Lock == 1 )
    {
        Returned = 1;
    }
    else if ( Unlock == 1 )
    {
        Returned = 0;
    }
    else
    {
        Returned = -1;
    }
}
```

PasswordTimeover()

```
int PasswordTimeover()
{
    if ( PowerStatus == false )
    {
        return -1;
    }

    if ( AlarmPW == true )
    {
        return 3;
    }
    else
    {
        return -1;
    }
}
```

```
void PasswordTimeover(int& Returned, bool PowerStatus_I, bool AlarmPW_I)
{
    if ( PowerStatus_I == false )
    {
        Returned = -1;
    }

    if ( AlarmPW_I == true )
    {
        Returned = 3;
    }
    else
    {
        Returned = -1;
    }
}
```

ClosedSensorLock()

```
int ClosedSensorLock()
{
    if (AlarmCS == true)
    {
        return 1;
    }
    else
    {
        return -1;
    }
}
```

```
void ClosedSensorLock(int& Returned, bool AlarmCS_I)
{
    if (AlarmCS_I == true)
    {
        Returned = 1;
    }
    else
    {
        Returned = -1;
    }
}
```

NumberCheck()

```
int NumberCheck(bool lu)
{
    static int result;

    int *Input = new int[4];

    if ( lu == false )
    {
        return result;
    }

    NumberRecord(&Input);

    if ( Input == NULL )    // 입력이 없으면 -1 리턴
    {
        result = -1;
        return -1;
    }
    for (int i = 0; i < 4; i++)
    {
        if ( Saved[i] != Input[i] )
        {
            StartNB = -1;   // 불합격이므로 알람 제거
            result = 2;
            return 2;
        }
    }

    StartNB = -1;   // 합격이므로 알람 제거
    result = 1;
    return 1;
}
```

```
void NumberCheck(int& Returned, int* Saved_S, bool& TimerNB, int* Input_I)
{
    if ( Input_I == NULL )  // 입력이 없으면 -1 리턴
    {
        Returned = -1;
        return;
    }
    for (int i = 0; i < 4; i++)
    {
        if ( Saved_S[i] != Input_I[i] )
        {
            TimerNB = false;    // 불합격이므로 알람 제거
            Returned = 2;
            return;
        }
    }
    TimerNB = false;    // 합격이므로 알람 제거
    Returned = 1;
}
```

ClosedSensorOpened()

```
void ClosedSensorOpened()
{
    if ( ClosedSensorReceiver() == 1 )
    {
        StartCS = -1;
    }
}
```

```
void ClosedSensorOpened(bool& TimerCS, int Input_I)
{
    int Input = Input_I;

    if ( Input == 1 )
    {
        TimerCS = false;
    }
}
```

ClosedSensorClosed()

```
void ClosedSensorClosed()
{
    if ( ClosedSensorReceiver() == 0)
    {
        if ( StartCS == -1 )
        {
            TIMER(TimerCS); // TimerCS 스레드 작동
        }
        else
        {
            StartCS = clock();
        }
    }
}
```

```
void ClosedSensorClosed(bool& TimerCS, int Input_I)
{
    int Input = Input_I;

    if ( Input == 1 )
    {
        TimerCS = true;
    }
}
```

PasswordSetup()

```
int PasswordSetup()
{
    static int Temp[4] = {-1, -1, -1, -1};

    int Input = InputFilter(true, 3);

    if ( Input >= 0 && Input <= 9 ) // 입력받은 값이 0 ~ 9 일 경우
    {
        if ( StartPW == -1 )
        {
            TIMER(TimerPW); // TimerPW 스레드 작동
        }
        else
        {
            StartPW = clock();
        }

        if ( AlarmPW == true )  // 알람이 울렸을 경우 초기화하고 버튼눌림 알림
        {
            for (int i = 0; i < 4; i++)
            {
                Temp[i] = -1;
            }
            return 1;
        }
        else if ( Temp[0] == -1 )    // 임시공간 첫번째칸이 비었을 경우
        {
            Temp[0] = Input;
            return 1;
        }
        else if ( Temp[1] == -1 )    // 임시공간 두번째칸이 비었을 경우
        {
            Temp[1] = Input;
            return 1;
        }
        else if ( Temp[2] == -1 )    // 임시공간 세번째칸이 비었을 경우
        {
            Temp[2] = Input;
            return 1;
        }
        else if ( Temp[3] == -1 )    // 임시공간 네번째칸이 비었을 경우 저장하고 초기화
        {
            Temp[3] = Input;

            int Result[4];

            for (int i = 0; i < 4; i++)
            {
                Result[i] = Temp[i];
                Temp[i] = -1;
            }

            PasswordSave(Result);    // 패스워드 저장

            return 1;
        }
    }

    if ( AlarmPW == true )  // 알람이 울렸을 경우 초기화
    {
        for (int i = 0; i < 4; i++)
        {
            Temp[i] = -1;
        }
    }
    return -1;
}
```

PasswordSetup()

```
void PasswordSetup(int& Returned, int& Activated, bool& TimerPW, int* Temp_S, int Input_I, bool AlarmPW_I)
{
    int Input = Input_I;

    if ( Input >= 0 && Input <= 9 ) // 입력받은 값이 0 ~ 9 일 경우
    {
        TimerPW = true; // PW타이머 작동

        if ( AlarmPW_I == true )    // 알람이 울렸을 경우 초기화하고 버튼눌림 알림
        {
            for (int i = 0; i < 4; i++)
            {
                Temp_S[i] = -1;
            }
            Returned = 1;
        }
        else if ( Temp_S[0] == -1 ) // 임시공간 첫번째칸이 비었을 경우
        {
            Temp_S[0] = Input;
            Returned = 1;
        }
        else if ( Temp_S[1] == -1 ) // 임시공간 두번째칸이 비었을 경우
        {
            Temp_S[1] = Input;
            Returned = 1;
        }
        else if ( Temp_S[2] == -1 ) // 임시공간 세번째칸이 비었을 경우
        {
            Temp_S[2] = Input;
            Returned = 1;
        }
        else if ( Temp_S[3] == -1 ) // 임시공간 네번째칸이 비었을 경우 저장하고 초기화
        {
            Temp_S[3] = Input;
            Activated = 1;
            for (int i = 0; i < 4; i++)
            {
                Temp_S[i] = -1;
            }
            Returned = 1;
        }
    }
    else
    {
        if ( AlarmPW_I == true )    // 알람이 울렸을 경우 초기화
        {
            for (int i = 0; i < 4; i++)
            {
                Temp_S[i] = -1;
            }
        }
        Returned = -1;
    }
}
```

NumberRecord()

```
void NumberRecord(int** Result)
{
    static int Temp[4] = {-1, -1, -1, -1};

    int Input = NumberButtonReceiver(true);

    if ( Input >= 0 && Input <= 9 ) // 입력받은 값이 0 ~ 9 일 경우
    {
        if ( AlarmNB == true )  // 알람이 울렸을 경우 초기화하고 버튼눌림 알림
        {
            for (int i = 0; i < 4; i++)
            {
                Temp[i] = -1;
            }
            *Result = NULL;
            return;
        }
        else if ( Temp[0] == -1 )   // 임시공간 첫번째칸이 비었을 경우
        {
            Temp[0] = Input;
            *Result = NULL;
            return;
        }
        else if ( Temp[1] == -1 )   // 임시공간 두번째칸이 비었을 경우
        {
            Temp[1] = Input;
            *Result = NULL;
            return;
        }
        else if ( Temp[2] == -1 )   // 임시공간 세번째칸이 비었을 경우
        {
            Temp[2] = Input;
            *Result = NULL;
            return;
        }
        else if ( Temp[3] == -1 )   // 임시공간 네번째칸이 비었을 경우 저장하고 초기화
        {
            Temp[3] = Input;
            for (int i = 0; i < 4; i++) // 현재 임시목록을 제거
            {
                *(*Result+i) = Temp[i];
                Temp[i] = -1;
            }
            return;
        }
    }
    else
    {
        if ( AlarmNB == true )  // 알람이 울렸을 경우 초기화
        {
            for (int i = 0; i < 4; i++)
            {
                Temp[i] = -1;
            }
        }
        *Result = NULL;
    }
}
```

NumberRecord()

```
void NumberRecord(int& Returned, int* Temp_S, int Input_I, bool AlarmNB_I)
{
    int Input = Input_I;

    if ( Input >= 0 && Input <= 9 ) // 입력받은 값이 0 ~ 9 일 경우
    {
        if ( AlarmNB_I == true )    // 알람이 울렸을 경우 초기화하고 버튼눌림 알림
        {
            for (int i = 0; i < 4; i++)
            {
                Temp_S[i] = -1;
            }
            Returned = -1;
        }
        else if ( Temp_S[0] == -1 ) // 임시공간 첫번째칸이 비었을 경우
        {
            Temp_S[0] = Input;
            Returned = -1;
        }
        else if ( Temp_S[1] == -1 ) // 임시공간 두번째칸이 비었을 경우
        {
            Temp_S[1] = Input;
            Returned = -1;
        }
        else if ( Temp_S[2] == -1 ) // 임시공간 세번째칸이 비었을 경우
        {
            Temp_S[2] = Input;
            Returned = -1;
        }
        else if ( Temp_S[3] == -1 ) // 임시공간 네번째칸이 비었을 경우 저장하고 초기화
        {
            Temp_S[3] = Input;

            // 완성된 4자리 숫자를 다른 곳에 임시저장

            for (int i = 0; i < 4; i++) // 현재 임시목록을 제거
            {
                Temp_S[i] = -1;
            }
            Returned = 1;   // 다른 곳에 저장한 임시목록을 반환
        }
    }
    else
    {
        if ( AlarmNB_I == true )    // 알람이 울렸을 경우 초기화
        {
            for (int i = 0; i < 4; i++)
            {
                Temp_S[i] = -1;
            }
        }
        Returned = -1;
    }
}
```

InputFilter()

```
int InputFilter(bool pw, int sq_I)
{
    if ( PowerStatus == true )  // 암호초기화가 안 되어 있으면 무조건 숫자키버튼만 반환
    {
        if ( pw == false )
        {
            return -1;
        }

        if ( sq == -1 )
        {
            return NumberButtonInput;
        }
        else
        {
            return -1;
        }
    }
    else
    {
        if ( sq == -1)  // 초기상태(sq == -1)에서만 센서인식 수행
        {
            sq = sq_I;  // 센서인식 수행함을 알림

            InputData[0] = ClosedSensorInput;
            ClosedSensorInput = -1;
            InputData[1] = ManualLockInput;
            ManualLockInput = -1;
            InputData[2] = KeyInput;
            KeyInput = -1;
            InputData[3] = NumberButtonInput;
            NumberButtonInput = -1;
            InputData[4] = CoverInput;
            CoverInput = -1;
        }
        else
        {
            sq = sq_I;
        }

        return InputData[sq];
    }
}
```

InputFilter()

```
void InputFilter(int& Returned, int* InputData_S, int& sq_S, bool& PowerStatus_S, int* Input_I, int& sq_I)
{
    if ( PowerStatus_S == true )    // 암호초기화가 안 되어 있으면 무조건 숫자키버튼만 반환
    {
        if ( sq_S == -1 )
        {
            Returned = Input_I[3];
        }
        else
        {
            Returned = -1;
        }
    }
    else
    {
        if ( sq_S == -1)    // 초기상태(sq == -1)에서만 센서인식 수행
        {
            sq_S = sq_I;    // 센서인식 수행함을 알림

            for (int i = 0; i < 5; i++) // 센서인식 수행
            {
                InputData_S[0] = Input_I[0];
                InputData_S[1] = Input_I[1];
                InputData_S[2] = Input_I[2];
                InputData_S[3] = Input_I[3];
                InputData_S[4] = Input_I[4];
            }
        }
        else
        {
            sq_S = sq_I;
        }
        Returned = InputData_S[sq_S];
    }
}
```

EDLS_UTC_000_000()

```
void EDLS_UTC_000_000()
{
    // 수행결과
    int Returned = -1;

    // 상태조건
    int InputData_S[5] = {-1, -1, -1, -1, -1};
    int sq_S = -1;
    bool PowerStatus_S = false;

    // 입력
    int sq_I = 0;
    int Input_I[5] = {0, 1, 2, 3, 4};

    // 기대값
    int Returned_E = 0;
    int InputData_E[5] = {0, 1, 2, 3, 4};
    int sq_E = 0;

    // 테스팅 결과
    InputFilter(Returned, InputData_S, sq_S, PowerStatus_S, Input_I, sq_I);
    Assertion("EDLS_UTC_000_000", Returned, InputData_S, sq_S, Returned_E, InputData_E, sq_E);
}
```

EDLS_UTC_017_000()

```
void EDLS_UTC_017_000()
{
    // 수행결과
    int Saved[4] = {-1, -1, -1, -1};

    // 상태조건
    bool PowerStatus_S = true;
    bool TimerPW_S = true;

    // 입력
    int Input_I[4] = {3, 5, 7, 8};

    // 기대값
    int Saved_E[4] = {3, 5, 7, 8};
    bool PowerStatus_E = false;
    bool TimerPW_E = false;

    // 테스팅 결과
    PasswordSave(Saved, PowerStatus_S, TimerPW_S, Input_I);
    Assertion("EDLS_UTC_017_000", Saved, PowerStatus_S, TimerPW_S, Saved_E, PowerStatus_E, TimerPW_E);
}
```

Result

```
EDLS_UTC_000_000 : Pass
EDLS_UTC_000_001 : Pass
EDLS_UTC_000_002 : Pass
EDLS_UTC_000_003 : Pass
EDLS_UTC_000_004 : Pass
EDLS_UTC_000_005 : Pass
EDLS_UTC_000_006 : Pass
EDLS_UTC_000_007 : Pass

EDLS_UTC_001_000 : Pass
EDLS_UTC_001_001 : Pass
EDLS_UTC_001_002 : Pass
EDLS_UTC_001_003 : Pass
EDLS_UTC_001_004 : Pass

EDLS_UTC_002_000 : Pass
EDLS_UTC_002_001 : Pass
EDLS_UTC_002_002 : Pass
EDLS_UTC_002_003 : Pass

EDLS_UTC_003_000 : Pass
EDLS_UTC_003_001 : Pass
EDLS_UTC_003_002 : Pass
EDLS_UTC_003_003 : Pass

EDLS_UTC_004_000 : Pass
EDLS_UTC_004_001 : Pass
EDLS_UTC_004_002 : Pass
EDLS_UTC_004_003 : Pass
EDLS_UTC_004_004 : Pass
EDLS_UTC_004_005 : Pass
```

```
EDLS_UTC_005_000 : Pass
EDLS_UTC_005_001 : Pass
EDLS_UTC_005_002 : Pass
EDLS_UTC_005_003 : Pass
EDLS_UTC_005_004 : Pass

EDLS_UTC_006_001 : Pass
EDLS_UTC_006_002 : Pass
EDLS_UTC_006_003 : Pass
EDLS_UTC_006_004 : Pass
EDLS_UTC_006_005 : Pass
EDLS_UTC_006_006 : Pass
EDLS_UTC_006_007 : Pass
EDLS_UTC_006_008 : Pass
EDLS_UTC_006_009 : Pass

EDLS_UTC_007_000 : Pass
EDLS_UTC_007_001 : Pass

EDLS_UTC_008_000 : Pass
EDLS_UTC_008_001 : Pass

EDLS_UTC_010_000 : Pass
EDLS_UTC_010_001 : Pass
EDLS_UTC_010_002 : Pass
EDLS_UTC_010_003 : Pass
EDLS_UTC_010_004 : Pass
EDLS_UTC_010_005 : Pass
EDLS_UTC_010_006 : Pass
```

```
EDLS_UTC_011_000 : Pass
EDLS_UTC_011_001 : Pass

EDLS_UTC_012_000 : Pass
EDLS_UTC_012_001 : Pass

EDLS_UTC_013_000 : Pass
EDLS_UTC_013_001 : Pass
EDLS_UTC_013_002 : Pass

EDLS_UTC_014_000 : Pass
EDLS_UTC_014_001 : Pass

EDLS_UTC_015_000 : Pass
EDLS_UTC_015_001 : Pass

EDLS_UTC_016_000 : Pass
EDLS_UTC_016_001 : Pass
EDLS_UTC_016_002 : Pass
EDLS_UTC_016_003 : Pass

EDLS_UTC_017_000 : Pass
EDLS_UTC_017_001 : Pass

EDLS_UTC_018_000 : Pass
EDLS_UTC_018_001 : Pass
EDLS_UTC_018_002 : Pass
EDLS_UTC_018_003 : Pass

EDLS_UTC_019_000 : Pass
EDLS_UTC_019_001 : Pass
EDLS_UTC_019_002 : Pass
EDLS_UTC_019_003 : Pass
```

```
EDLS_UTC_020_000 : Pass
EDLS_UTC_020_001 : Pass
EDLS_UTC_020_002 : Pass
EDLS_UTC_020_003 : Pass

EDLS_UTC_021_000 : Pass
EDLS_UTC_021_001 : Pass
EDLS_UTC_021_002 : Pass
EDLS_UTC_021_003 : Pass

EDLS_UTC_022_000 : Pass
EDLS_UTC_022_001 : Pass
EDLS_UTC_022_002 : Pass
EDLS_UTC_022_003 : Pass

EDLS_UTC_023_000 : Pass
EDLS_UTC_023_001 : Pass
EDLS_UTC_023_002 : Pass
EDLS_UTC_023_003 : Pass

EDLS_UTC_024_000 : Pass
EDLS_UTC_024_001 : Pass
EDLS_UTC_024_002 : Pass

EDLS_UTC_025_000 : Pass
EDLS_UTC_025_001 : Pass
EDLS_UTC_025_002 : Pass
EDLS_UTC_025_003 : Pass

EDLS_UTC_026_000 : Pass
EDLS_UTC_026_001 : Pass
EDLS_UTC_026_002 : Pass
```