

Documentation About Methods Used In [Plan_it] Project

Authors: 2023011393 Nawon Kim, 2021114026 Jeongwoo Kim, 2023013565 Dahye Jeong

Version 0.0.2 이후의 버전부터 추가되거나 수정된 사항만 반영되어 있습니다.

0.0.3 수정사항

(Core)

삽입시 기존에 북마크가 있을 경우 새 자료 무시

윤년, 28, 29, 30, 31일 구분 및 잘못된 날짜를 보유한 데이터는 처음부터 생성되지 않도록 조치

human readable 파일 생성시, 3년의 인터벌을 두는 범위(2023-2025 등)의 경우 첫 연도의 트리가 잘못 형성되는 오류를 제거

해시값 구성 조건 새로 구성

(UX)

좌표 공간 새로 구성

하이라이트 기능 수정

0.0.5 수정사항

(Core) 레코드 삭제 기능 구현

(Core) 반복 세이브/로드시 메모리 누수 문제 해결, 레코드 필드에 대한 동적 할당 해제 기능 적용

(Core) getTodaySchedule_계열 함수 반환형 int로 업데이트, 반환 데이터가 의미를 가짐.

(Core) sighandler 및 itimer 기반 리마인더 기능 구현

(Core) 리마인더 세이브/로드 기능 구현(저장 파일과 함께 저장됨), Human-readable 파일 내에도 Reminder 정보 제공

(Core) 현재 실행 중인 리마인더 삭제 / 확인 기능 구현

(Core) 실행시 Core부 초기화 함수 coreInit() 내부적 업데이트

(Core) 일정 수정 기능 소급 적용

(UX) Core부 함수 참조 부분에 대해 반환형 개선 및 신규 함수 추가

0.0.6 수정사항

(Core) UX 화면 뿐만 아니라 -d 옵션으로 진입 가능한 디버깅 메뉴에서도 띄어쓰기가 포함된 문자열 입력 가능

(Core) 수정 기능 관련 함수 구현 중

(Core) 양방향 iterator 기반 상세 일정 순회 기능으로 확장

(UX) 일정 세부사항 조회 기능 추가

(UX) 일정 세부사항 조회에 양방향 iterator 기반 양방향 순회 추가

(UX) 문자열 길이 측정 관련 GDB 기반 오류 현재 파악, 해결 위해 노력 중 (아래 별첨 참고)

(기타) 랜덤 레코드 생성 프로그램 내부 랜덤 문자열 업데이트, 세이브파일 업데이트

0.0.7 수정사항

(Core) 띄어쓰기가 포함된 문자열 기능 안정화

(Core) 동적 할당 해제 알고리즘 개선

(Core) UX레이어와의 상호작용을 위해 일부 함수 내부 구조 개선

(UX) 메뉴 심도 3단계 -> 2단계로 단순화

(UX) 양방향 순회 방식 일자 탐색, 선택되어 있는 대상을 바로바로 지울 수 있도록 개선 (메뉴 심도의 단축과도 관련 있는 기능)

(UX) 삽입 함수를 낮춰진 심도인 2단계에 삽입

(UX) 일정 개수 출력 함수의 string refresh 관련 알고리즘 개선

(UX) details string parser 알고리즘 개선

0.0.9 수정사항

(Server) 초대코드 및 마크업 스트링 기반의 공유 시스템 구현, proof-of-concept 단계.

(Client) 초대코드 및 마크업 스트링 기반의 공유 시스템 구현, proof-of-concept 단계.

(Core) 동적 할당 해제 알고리즘 일부 개선

(Core) default 레코드 생성 프로그램 개선, 편의성 제고

(Core) 추상화를 거치지 않고 UX에서 활용되는 함수들의 오류 수정, 각주 추가

(UX) 색상별 북마크 스티커의 구현 및 달력/우측 화면에의 표시

(UX) 요일별 일자 표시 색 적용, 각종 음영 및 색 조합 관련 개선으로 가독성 향상

(UX) 공휴일 적색으로 표시, 공휴일 관련 DB 함수 완전 적용 완료

(UX) Reminder의 UX-Level 구현

(UX) getCommandScreen() 범용 함수 구현

Week 2 수정사항

UX 파트 (plnit_uxCore.c)

void clearGivenCalendarArea(int row, int col);

⇒ 6 by 7 구조의 달력의 각 칸을 지정하여, 그 칸의 모든 내용을 지웁니다.

void clearGivenRowCols(int fromRow, int fromCol, int toRow, int toCol);

⇒ clearGivenNonCalendarArea에서 내부적으로 사용되는 함수입니다.

void clearGivenNonCalendarArea(/*pre-defined Macros*/int area);

⇒ SLL, SLC 등의 영역을 매크로(#define)로 지정하였기에,

clearGivenNonCalendarArea(SLC) 등의 방식으로 사용 가능합니다.

void save_UXPart(void);

⇒ 저장되었음을 알리는 UX적인 기능을 추가적으로 발휘하며, 저장 액션을 수행하는 함수입니다.

Critical part인 만큼, sigprocmask등으로 시그널 블로킹이 필요 해 보입니다.

void load_UXPart(void);

⇒ 로드되었음을 알리는 UX적인 기능을 추가적으로 발휘하며, 로드 액션을 수행하는 함수입니다.

Critical part인 만큼, sigprocmask등으로 시그널 블로킹이 필요 해 보입니다.

void setInputModeSigHandler(int status);

⇒ 입력 모드 수행 중 나가기 동작을 수행하기 위한 signal handling method입니다.

void inputMode_sigHndl(int signum);

⇒ 입력 모드용 signal handler 입니다.

void popup(char* title, char* str1, char* str2, int delay)

⇒ title에는 알림의 제목을, str1과 str2는 알림의 내용을 표시 할 수 있습니다. Delay는 초 단위로 입력받으며, 현재 프로세스에 미리 알림 등으로 인해 itimer가 설정 되었을 경우, 충돌을 피하기 위해 fork()로 구현되어 있습니다.

void edit_plan(unsigned long long targetDate, int* page);

⇒ ??? int* page?

Core 파트 (plnit_dbCore.c)

void coreInit(void)

⇒ UX 엔진 부팅시 DB 엔진 초기화 함수입니다.

void getBookMarkedInDate(unsigned long long today, int counter, char* str)

- ⇒ UX부를 위한 북마크 스트링 추상화 함수입니다. toDoPtr getBookMarked(unsigned long long src, int distance) 및 해시함수를 기반으로 작동합니다. counter에는 오늘을 기준으로 앞으로 몇 개의 북마크를 보여줄 것인지에 해당하는 데이터가 들어갑니다.
- ⇒ *[^15:30]^Sample_Title: 북마크 된, 시간이 15:30인, 제목이 Sample_Title인 스트링
- ⇒ 북마크 된 것만 불러오는데, 북마크 prefix code가 들어간 이유는 아래 format string 코드 재활용을 하여 작성이 수월하도록, 도와드리기 위함입니다.

void deleteWhileIterate(unsigned long long src, int pageNum);

- ⇒ YYYYMMDD식의 src의 iterable page number을 기준으로(주의: index 아님) 삭제합니다.

int editWhileIterate(unsigned long long src, int pageNum);

- ⇒ YYYYMMDD식의 src의 iterable page number을 기준으로(주의: index 아님) 수정합니다. 다만 아직 미구현 상태입니다.

int setReminder(time_t current, time_t delta, int repeatCnter, char* what, int intervals);

- ⇒ Current는 time(NULL)을 넣으며(저장 기능의 용이성을 위해 있는 인자라서, 그냥 활용하실 때는 time(NULL)을 넣으시면 됩니다.), delta는 현재 기준으로 n초 후의 미리 알림을 설정하겠다는 뜻이며, repeatCnter과 intervals는 미리 알림 종료 시점 기준으로 얼마나 전부터 몇번 미리 알림을 띄우겠다 라는 의미입니다.
- ⇒ 즉, 지금이 1시고 2시에 있는 약속을 30분 전부터 3번 미리 알림을 띄우고자 하면, 1시 30분, 1시 40분, 1시 50분에 미리 알림이 뜰 것입니다.
- ⇒ What은 내용 string이며, 30자 제한입니다.

int isReminderSetAlready(char* str);

- ⇒ 리마인더가 설정되어 있다면, 1을 반환함과 동시에 str에 앞서 말한 What이 저장됩니다.
- ⇒ 라마인더가 설정되어 있지 않으면, 0을 반환합니다.

void turnOffReminder(void);

- ⇒ 호출시 리마인더를 끕니다.

int getTodaySchedule_Summarized(unsigned long long today, char* strbuf);

- ⇒ 업데이트 되었습니다. 삭제 기능이 구현된 만큼, 해당 자리에 레코드가 존재하지 않는다면 -1을 리턴하고 buf에 "no_data"를 기록합니다.
- ⇒ 레코드가 있다면, 0을 반환합니다.

int getTodaySchedule_withDetails(unsigned long long today, char* strbuf);

- ⇒ 업데이트 되었습니다. 삭제 기능이 구현된 만큼, 해당 자리에 레코드가 존재하지 않는다면 -1을 리턴하고 buf에 "no_data"를 기록합니다.
- ⇒ 레코드가 있다면, 0을 반환합니다.

int setSchedule(unsigned long long today, char* title, char* details, int priority)

- ⇒ 정상적으로 삽입이 완료되면 0
- ⇒ 북마크가 겹치면 1
- ⇒ 유효하지 않은 날짜는 2를 반환합니다.

void reminderHandler(int signum)

- ⇒ 기능 구현 여부를 보이기 위해 CLI상에서 구동되지만, 실제로는 UX에서 이루어져야 하는 시그널 핸들링입니다. 구현 방식과 가이드라인은 dbCore 파일 내 각주를 참고해 주시면 감사하겠습니다.

Format string related

- ⇒ 조언 해 주신대로 prefix code를 단순화 하여서, time, title, detail, bookmark 여부를 다음과 같은 논리에 기반하여 format string을 리턴합니다. (new line 여부는 ux단에서 해결합니다)
- ⇒ [^15:30*]^Sample_Title: 북마크* 된, 시간이 15:30인, 제목이 Sample_Title인 스트링
- ⇒ [^15:30]^Sample_Title: 시간이 15:30인, 제목이 Sample_Title인 스트링
- ⇒ [^15:30*]^%Sample_Title]]Sample_Details: 북마크* 된, 시간이 15:30이고, 제목이 Sample_Title이며, 내용이 Sample_Details
- ⇒ [^15:30]^%Sample_Title]]Sample_Details: 시간이 15:30이고, 제목이 Sample_Title이며, 내용이 Sample_Details

int editWhileIterate(unsigned long long src, int pageNum, unsigned long long t_day, char* t_title, char* t_details, int t_priority) => 폐기

- ⇒ 내부적으로 삭제 후 편집 대상 삽입의 방식으로 구현되어 있습니다. 단, 과정이 아래와 같습니다.
- ⇒ 편집 대상 페이지가 존재하지 않는 경우: 2 리턴
- ⇒ 편집을 시도하려고 하였으나, 편집 대상이 새로 북마크된 속성을 가지도록 하였는데 이미 그 일자에 북마크된 일정이 존재할 때: 1 리턴
- ⇒ 정상적으로 "삭제 후 삽입"이 일어났을 때: 0 리턴
- ⇒ 기본적으로 deleteWhileIterate에 정보 추가 기능이 있다고 생각하시면 되고, 그러한 사고방식으로 함수 인자에 접근하시면 됩니다.
- ⇒ setSchedule, editWhileIterate 등 리턴값으로 오류 정보를 알아낼 수 있는 경우 적절한 텍스트로 핸들링 해 주시면 감사하겠습니다.

int getTodaySchedule_withDetails(unsigned long long today, char* strbuf, int direction)

- ⇒ 순회 방향 지정 가능하도록 direction 인자 추가

int editWhileIterate(unsigned long long src, int pageNum, unsigned long long t_day, char* t_title, char* t_details, int t_priority)

- ⇒ Src와 pagenum은 __whileIterate() 시리즈와 동일하나, 자료를 새로 만들때와 동일한 수준의 인자인 day, title, details. Bookmark no.를 필요로 합니다. 삭제에 기반한 수정이 아니라 실제로 메모리를 수정하도록 기능합니다. 단, 유저의 입력에 따라 수정을 원치 않는 부분도 있을 텐데 그것은 아래와 같이 구분합니다.

- ⇒ t_day를 1234로 주면 YYYYMMDDHHMM 수정은 없음
- ⇒ (중요)title[0]이 NULL이면(즉 null 포인터가 아닌 문자열 배열인데 첫 칸이 널인) title 수정 없음
- ⇒ (중요)details[0]이 NULL이면 details 수정 없음
- ⇒ Bookmark(t_priority)가 -1이면 북마크 여부 수정은 없음
- ⇒ 리턴값은 정상 수행시 0, 지울게 없으면 2, 북마크 충돌시(즉 기존에 북마크가 존재하는데 수정을 통해 하나 더 생성하려고 할 시) 1입니다.

char getCommandScreen(char* context, char availToken[])

- ⇒ Context에 스트링으로 된 질의 내용을, avail 토큰에는 "yYnN"과 같이 유효 입력을 검사하는 스트링을 인자로 넣습니다.
- ⇒ 올바른 입력이 들어오면(유효 입력 문자 내의 문자) 해당 문자를 반환합니다.

void printReminderControl(int how)

- ⇒ how에는 SET, DEL 상수가 올 수 있습니다.
- ⇒ SET은 리마인더를 설정하거나 덮어쓰며, DEL은 리마인더를 질의 후 삭제합니다.

void reminder_extends_popup(int signum)

- ⇒ UX 레이어용 리마인더 시그널 핸들링 함수입니다.

<동적 할당 관련 업데이트> :: pre-0.1.0

```
==7120==
==7120== HEAP SUMMARY:
==7120==    in use at exit: 0 bytes in 0 blocks
==7120==   total heap usage: 28,189 allocs, 28,189 frees, 51,095,704 bytes allocated
==7120==
==7120== All heap blocks were freed -- no leaks are possible
==7120==
==7120== Use --track-origins=yes to see where uninitialised values come from
==7120== For lists of detected and suppressed errors, rerun with: -s
==7120== ERROR SUMMARY: 10 errors from 1 contexts (suppressed: 0 from 0)
jeongwoo_kim@jeongwoo:~/synoGo/openToDo$
```

Save시 문제, D-day 관련 문제가, Stack trace 결과 모두 동적 할당 메모리 소유권 문제에서 비롯 된 것 같아, 한번 빠르게 잡았습니다. dbCore 부분에만 해당되는 결과입니다.

이걸로 문제가 많이 잡혔으니, UX파트 분들도 시도해 보시면 좋을 것 같습니다.(2025년 이후 문제 등)

<D-day 관련 함수 업데이트> :: pre-0.1.0

[API]

```
void getDday(int* slot1, char* title1, int* slot2, char* title2);
```

- ⇒ 모든 인자는 호출단에서 정적/동적으로 할당되어 있어야 합니다. 즉, 특히 스트링 같은 경우, 메모리를 할당하여 넘겨주는 것이 아닌, 주어진 버퍼에 작성해서 넘겨주는 구조입니다.
- ⇒ (slotN, titleN)에서 slot은 D+- 일자 수를 담아주는 역할이고, 따라서 포인터 변수입니다. Title은 d-day 제목을 담아주는 역할이고, 길이는 DB상의 Title을 그대로 가져옵니다.
- ⇒ (slot1, title1)은 D+ 데이터를 담고 있습니다. 데이터가 존재하지 않을 시 title1[0]에 널 문자가 삽입됩니다. %+d 형식 지정자를 통해 부호를 포함하여 출력하면, slot의 데이터를 dday화 합니다.
- ⇒ (slot1, title1)은 D- 데이터를 담고 있습니다. 데이터가 존재하지 않을 시 title1[0]에 널 문자가 삽입됩니다. %+d 형식 지정자를 통해 부호를 포함하여 출력하면, slot의 데이터를 dday화 합니다.

```
void setDdayWhileIterate(unsigned long long src, int pageNum, int mode);
```

- ⇒ 구조는 이터레이터 기반 출력을 다루던 때와 완전히 동일하며(YYYYMMDD), mode 매개변수가 추가되었습니다. 즉 detail 모드에서 보여지는 일정을 바로 D-day화 합니다.
- ⇒ Mode := 0이면 D+ 설정, 1이면 D- 설정입니다.
- ⇒ 따라서, 현재 일자보다 이전/이후에 커서가 있으냐에 따라 커맨드 창에서 D+- 중 하나를 아예 없애서 보여주지 않고, 입력을 무시함으로써 오류를 방지하는テクニック이 필요해 보입니다.

```
int checkDdayWhileIterate(unsigned long long src, int pageNum, int mode);
```

- ⇒ setDdayWhileIterate보다 선행되어야 합니다. (매우 중요)
- ⇒ 인자 구조는 선행 함수와 동일합니다.
- ⇒ 단, 반환값에 따라 다르게 대응해야 합니다:
 - 1 반환: 같은 일정(일자, 제목 동일)을 선택한 경우
 - 2 반환: 덮어쓰기가 발생할 경우
 - 0 반환: 바로 정상 입력(기준에 해당 슬롯을 차지하던 값이 없음)
 - ◆ 특히 1 반환시 사용자가 삭제를 원하면 delDdayStack(mode)로 삭제 해야 합니다.

[Core] :: Core에서 구현 된 방법에 대한 궁금증이 있으면 보시기 바랍니다.

```
void saveDday(int* fd);
```

```
void loadDday(int* fd);
```

```
void initDday(void);
```

```
int isBothDdayTheSame(todoPtr s, todo o);
```

⇒ 인자 구조가 특이한데, 코드에 설명이 잘 되어 있습니다.

```
void delDdayStack(int whatto);
```

⇒ 동일 자료 선택시 삭제 기능을 구현하는데 사용됩니다.

```
void setDdayStack(todoPtr target, int addto);
```

⇒ D-day 설정 구현 함수입니다.

<DB 구조 개편> :: pre-0.1.0

```
typedef struct todo {
    char userName[16];
    int isShared; /* 1: shared 2: being shared */
    char code[9]; /* if it's being shared */

    unsigned long long hashNum;
    unsigned long long dateData;
    int priority;
    char title[26];
    // SOCKET FEATURE {method}();
    char details[61];
} todo;
typedef todo* todoPtr;
typedef struct day {
    int isBookMarkExists;
    int sharedToDoExists;
    int maxIndex;
    int isHoliday;
    todoPtr* todoArr;
} day;
typedef day* dayPtr;
```

⇒ todo 에서 username, isShared, code 가 추가되었습니다.

⇒ userName 은 사용자 이름이거나 공유자의 이름 일 수 있습니다.

⇒ Code 는 공유 코드를 저장하는데, 주 용도는 "본인이 공유했을 때" 이를 지속적으로 사용할 수 있도록 하기 위함입니다. (기본값: -----)

⇒ isShared 가 중요한데, 1 이면 공유 '받은' 것, 2 이면 공유'한'것을 의미합니다. 다만 화면 표시에 사용되는 인자 '@'는 두 경우 모두 적용하는게 좋을 것 같습니다. 공유 일정 자체에 의미를 두는게 좋을 것 같기 때문입니다.

<Online Sharing Feature 관련 업데이트> :: pre-0.1.0

공유 기능으로 인해 DB 구조가 위와 같이 개편되어, 이전과 세이브 호환이 어렵습니다.

공휴일을 저장하는 시스템 디폴트 파일 내부 구조도 업데이트 되었습니다.

즉 public.dsv, todos.sv 모두 호환되지 않습니다.

[API]

```
void setUsername(char* myName);
```

⇒ 유저 이름을 설정하는 함수로, 아직 사용하지 않으셔도 됩니다.

⇒ 이렇게 있다는 정도만 아시고, 차후 최초 실행시를 구분하는 기능과 함께 사용할 예정입니다.

⇒ 기본 유저 이름은 Gildong_Hong 입니다.

```
int getFromServer_Highlevel(char* shareCode);
```

⇒ 공유 코드를 기반으로 서버에서 데이터를 불러와, 있다면 무조건 추가합니다.

⇒ 즉, 로컬 북마크와 공유 북마크가 겹칠 경우 로컬 북마크를 격하, 공유 북마크를 삽입합니다.

⇒ 토의에서는 "북마크가 겹칠 경우 삭제할 지 한번 더 질의"로 했는데, 그렇게 하지 않은 이유는:

- 1. 무언가를 공유 받았다는 의미가 그만큼 훨씬 더 중요한 일정이라는 의미입니다.
- 2. 그럼에도 기존에 본인이 가지고 있던 일정이 가지는 의미 또한 있을 것입니다.

⇒ 따라서, "공유 데이터가 북마크된 데이터 일 경우, 기존에 북마크 된 일정이 있었다면, 그 일정을 일반 일정으로 격하시키고 공유 데이터를 새로운 북마크로 추가합니다".

⇒ 리턴값: 1->코드에 맞는 데이터 없음, 2->서버 연결 불가, 0->정상

```
int shareWhileIterate(unsigned long long src, int pageNum, char* shareCode);
```

⇒ 앞서 많이 활용 되었던 whileIterate family 와 동일한 인자 구조(YYYYMMDD)이지만, shareCode 가 추가되었습니다.

⇒ 이 함수를 호출하면, 현재 페이지 일정을 공유하고, shareCode[9] '버퍼에 담아서' 줍니다.
내부에서 할당하여 주지 않습니다.

⇒ 리턴값: 3->DB 에 해당 데이터 존재하지 않음(안전성 향상용 기능) 2->서버 연결 불가 0->정상

```
int isSharedToDoExisting(unsigned long long targetDate);
```

⇒ YYYYMMDD 일자에 공유 되거나 공유 한 일정이 있으면 1 을 반환하고, 그렇지 않다면 0 을 반환합니다.

<빌드 방식 관련 업데이트> :: pre-0.1.0

⇒ 클라이언트 구현 파일까지 존재하여 빌드 식이 길어졌습니다:

- gcc -Wall -o pln plnit_clientCore.c plnit_dbCore.c plnit_uxCore.c -g

⇒ 따라서, 배치 파일을 만들었고, "./bp.sh"로 한번에 컴파일 및 오류 발생시 디스플레이 됩니다.

[API 업데이트]

int getTodaySchedule_Summarized(unsigned long long today, char* strbuf)

⇒ 내부를 직접 보시면서 이해하시면 좋을 것 같습니다.

```
⇒ if ((dd->toDoArr)[i]->isShared) {
⇒     sprintf(temp, "[%04llu*%d@]^%-25s", (dd->toDoArr)[i]->dateData %
10000, (dd->toDoArr)[i]->priority, (dd->toDoArr)[i]->title);
⇒     }
⇒     else if ((dd->toDoArr)[i]->priority) /* if bookmarked */ {
⇒         /*          Time->BookMarked->Title*/
⇒         sprintf(temp, "[%04llu*%d]^%-25s", (dd->toDoArr)[i]->dateData %
10000, (dd->toDoArr)[i]->priority, (dd->toDoArr)[i]->title);
⇒     }
⇒     else {
⇒         /*          Time->Title*/
⇒         sprintf(temp, "[%04llu]^%-25s", (dd->toDoArr)[i]->dateData % 10000,
(dd->toDoArr)[i]->title);
⇒     }
```

⇒ 공유 하거나 공유로 받은 것에 상관없이, 공유 일정이라는 의미 자체, 즉 협업에 중점을 두므로 '공유 상태'로 취급하며, 만약 '공유 상태'라면 북마크 숫자 뒤에 '@'를 넣습니다.

int getTodaySchedule_withDetails(unsigned long long today, char* strbuf, int direction)

```
if ((dd->toDoArr)[i]->isShared == 1) { /* 공유 받은 */
    sprintf(temp, "[%04llu*%d@%-15s]^%-25s]]%s", (dd->toDoArr)[i]->dateData % 10000, (dd-
>toDoArr)[i]->priority, (dd->toDoArr)[i]->userName, (dd->toDoArr)[i]->title, (dd->toDoArr)[i]->details);
}
else if ((dd->toDoArr)[i]->isShared == 2) { /* 공유 해 주는 */
    sprintf(temp, "[%04llu*%d@@%-8s]^%-25s]]%s", (dd->toDoArr)[i]->dateData % 10000, (dd-
>toDoArr)[i]->priority, (dd->toDoArr)[i]->code, (dd->toDoArr)[i]->title, (dd->toDoArr)[i]->details);
}
else if ((dd->toDoArr)[i]->priority) { /* if bookmarked */
    /*          Time->BookMarked->Title(NL)->Details*/
    sprintf(temp, "[%04llu*%d]^%-25s]]%s", (dd->toDoArr)[i]->dateData % 10000, (dd->toDoArr)[i]-
>priority, (dd->toDoArr)[i]->title, (dd->toDoArr)[i]->details);
}
else {
    /*          Time->Title(NL)->Details*/
    sprintf(temp, "[%04llu]^%-25s]]%s", (dd->toDoArr)[i]->dateData % 10000, (dd->toDoArr)[i]-
>title, (dd->toDoArr)[i]->details);
}
```

⇒ 공유로 받은 일정

- 북마크 숫자 뒤에 '@-15s'로 이름을 기입합니다.

⇒ 공유 한 일정

- 북마크 숫자 뒤에 '@@-8s'로 코드를 기입합니다.

⇒ UX 에서 각 상태에 적응형으로 대응하기 위한 방법으로는, 16 길이 버퍼를 선언한 뒤에 빈 상태로 초기화 시켜 놓고, @ 또는 @@에 따라 오는 데이터를 복사 한 뒤, 출력을 다루는 부분에서 버퍼의 strlen 에 따라 Shared by: 라고 출력할지 My Code: 라고 출력할지 정하면 좋을 것 같습니다.

[Core]

```
void deBookmarkInDate(unsigned long long src);
```

⇒ 공유 데이터와 북마크가 겹치는 경우, 로컬 북마크의 격하를 위해 사용되는 내부 함수입니다.

```
int pushToServer(todoPtr o, char* shareCode);
```

⇒ Client API 래핑 함수입니다.

```
int getFromServer(todoPtr target, char* shareCode);
```

⇒ Client API 래핑 함수입니다.

[Week 6 수정사항]

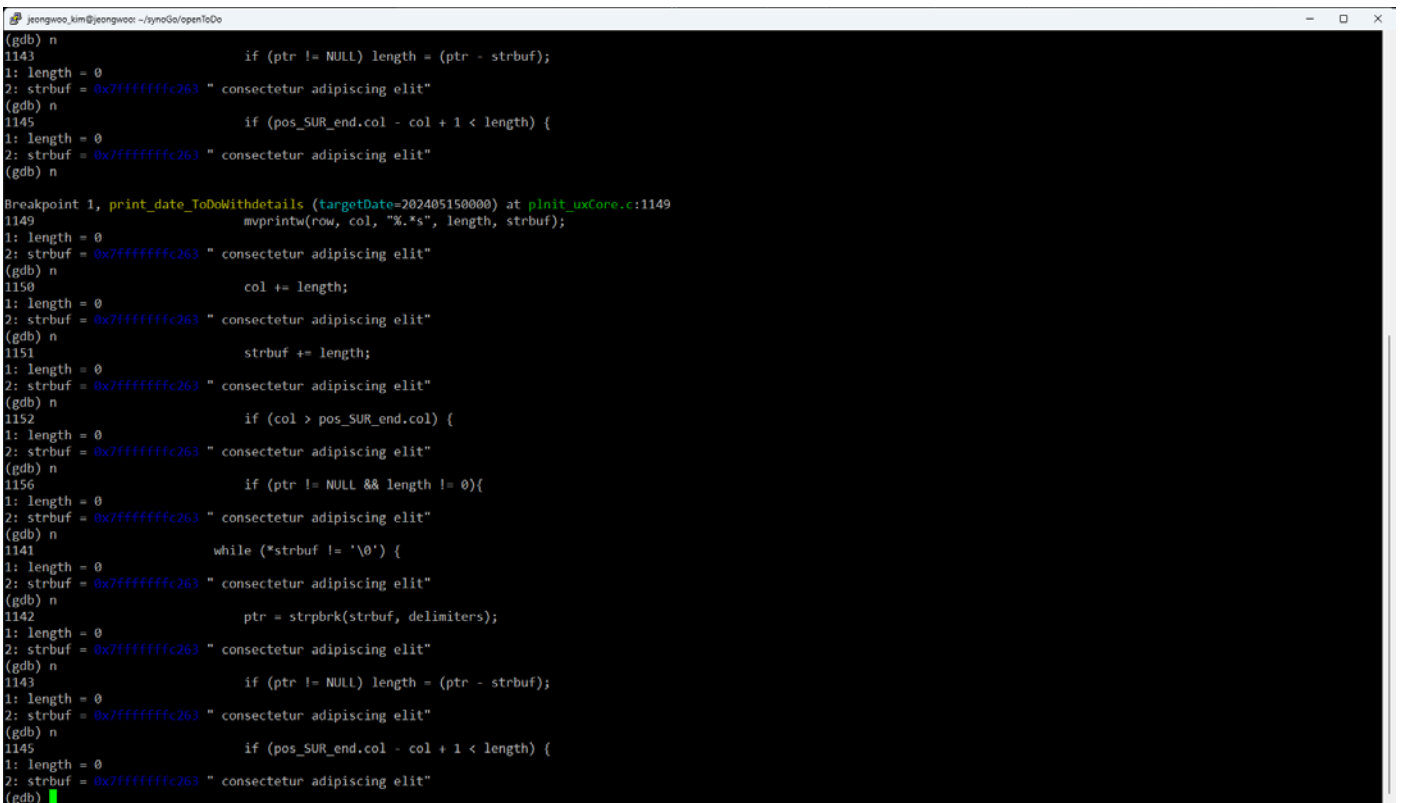
1. (API) void getDday(int* slot1, char* title1, int* slot2, char* title2)
 - A. 20 자 이상의 스트링은 15 자까지 표현되고, 뒤에 "..."이 붙는 것으로 바뀌었습니다.
 - B. 즉, UX 파트에서 n 이 최소일 때의 화면 여유분에 대응 할 수 있습니다.
2. (Core only) toDoPtr getBookMarked(unsigned long long src, int distance)
 - A. 이제 더 정밀한 탐색이 가능하도록, src 를 YYYYMMDDHHMM 으로 받습니다.
3. (API) getBookMarkedInDate(unsigned long long today, int counter, char* str)
 - A. 이제 더 정밀한 탐색이 가능하도록, src 를 YYYYMMDDHHMM 으로 받습니다.
 - B. 공유 일정에 대해서 추가적인 마크업 스트링을, UX 부로 발송합니다.
4. (UX) void print_UpcomingBookMark(unsigned long long today)
 - A. 현재 일자 기준으로, 다가오는 북마크를 연-월-일 | 시간:분의 양식으로 정확하게 안내합니다.
 - B. 다만, 공유 일정에 대한 마크업 스트링을 Core 에서 발송함에도 불구하고 별도로 표시되지 않습니다.
5. Client-Server Interaction 관련
 - A. Client Hello, Server Hello 및 buffer overrun/underrun 을 방지하는 기능의 추가로 더 안정적인 통신이 가능합니다.
6. UX Layer Input 개선
 - A. 북마크 스트립 지정 관련 직관성 향상 및 시간 관련 수정이 존재하지 않을 경우에 대한 반응성 향상이 있습니다.
7. (UX) void print_Dday() 개선
 - A. 하드코딩된 수치가 아닌, 스크린 사이즈를 결정하는 nNum 에 따라 동적으로 영역의 위치가 할당되도록 수정했습니다.
 - B. D-day 에 항목이 존재하지 않을 시, 별도의 메시지를 출력시켜 이러한 슬롯이 존재한다는 것을 유저에게 인지시킵니다.
8. (UX) void change_userName() 구현
 - A. 본인의 이름을 15 자 이하의 범위 내에서 변경 가능합니다.
 - B. 단, 변경된 이름의 상태가 저장되는 기능은 아직 미구현 상태입니다.
9. (Client) Server Connection 단계에서 getaddrinfo 적용
 - A. 안정적인 서버 연결을 위해, 도메인 기반으로 접속 요청시에 가장 효과적인 getaddrinfo() 라이브러리 함수를 적용하였습니다.
10. (DB) void getBookMarkedInDate(unsigned long long today, int counter, char* str) 업데이트
 - A. 현재 일자 기준으로, 이후의 일자에 존재하는 북마크의 실제 개수와 상관없이, 스크린 사이즈를 무조건 채우기 위해 같은 항목이 여러 번 출력되던 오류를 수정했습니다.
11. (DB) 메모리 안정성 강화
 - A. 서로 다른 라이브러리 간의 구조체 포인터 공유시 발생하는 문제를 주기적인 memset()으로 해결하였습니다.
 - B. 대표적으로, 공유 일정의 Title 에 잘못된 문자가 삽입되는 등의 Stack smashing 문제를 해결했습니다.

Week 3 수정사항

1. `int isBookMarked(unsigned long long targetDate);`
 - YYYYMMDD의 일자를 입력받아, 0(없음), 1, 2, 3은 지정된 색상에 따른 리턴 값을 제공합니다.
2. `int setDdayWhileIterate(unsigned long long src, int pageNum);`
 - 토의 된 대로, 페이지 열람 화면에서 현재 선택된 페이지에 있는 내용을 d-day의 기준 값으로 삼습니다. (memcpy 기반으로, 원본 레코드가 삭제되어도 남아있습니다.) 따라서 별도의 제목 입력, 날짜 입력이 필요 없어 직관적입니다.
 - 대상 데이터가 존재하지 않으면 2, 이미 2개의 d-day가 설정되어 있으면 1, 정상적으로 작동되었으면 0을 반환합니다. (알림창 요소로도 제공 가능)
 - D+, D- 구분을 별도로 받지 않습니다. 그냥 최대 2개를 저장 할 수 있는 구조입니다. (내부적으로 스택으로 구현되어 있어, 이론상 무한대의 개수를 저장할 수 있긴 하지만 입력단에서 2개로 제한을 걸었습니다.) getDday에서 더 자세하게 다루어집니다.
3. `void getDday(int* slot1, char* title1, int* slot2, char* title2);`
 - slot1, 2에는 정수형 변수의 주소를 넘겨주시면 됩니다. 그러면 D -+ 수를 채워서 드립니다.
 - Title 1, 2에는 title 최대 길이 만큼의 버퍼를 넘겨주시면 됩니다.
 - 데이터가 존재하지 않으면 title[0]에 NULL을 채워서 줍니다. 이를 통해 출력 유무를 판단하시면 됩니다.(slot은 그 어떠한 정수도 올 수 있으므로 slot으로 유무 판단은 어렵습니다)
 - 날짜의 차이를 윤년, 각 달의 마지막 일 수 까지 고려하여 정확한 일자의 차이를 구하는 알고리즘이 적용되어 있어, D+, D- 모두에 대응 할 수 있습니다. D+ 2개, D- 2개, D+ 및 D- 각각 하나씩 모두 경우의 수가 될 수 있습니다.
4. `void popDday(void);`
 - 저장된 d-day를 한 개씩 삭제합니다. 스택 구조라, LIFO임에 주의하셔서 프롬프트 메시지를 띄워주시면 좋을 것 같습니다.
5. `int isHoliday(unsigned long long target);`
 - YYYYMMDD의 일자를 입력받아, 공휴일이면 1, 일반적인 날(주말 및 평일 포함)이면 0을 반환합니다. 여담으로, 금주 회의에서 토의되었던 내용과는 달리 9의 북마크 값을 지니지 않고, 자료구조상의 개선을 통해(일자 구조체 포인터 관련 개선) 공휴일 여부를 저장합니다. 따라서, 공휴일 일정의 북마크 값은 9가 아닌 0이며, 시간은 토의된 대로 9999로 설정되어 있습니다.
 - 정렬 구조 개선으로, 9999 시간의(공휴일 일정)은 맨 첫 페이지에 오도록 되어 있습니다.
 - 9의 북마크를 지니지 않는 이유는, 내부 로직상 북마크 충돌 여부를 파악할 때 != 0 기반이라, 9로 설정할 시 공휴일에는 북마크 처리가 불가능해지기 때문입니다.
6. randomRecordGenerator의 샘플 개수를 20개로 늘리고, 타이틀 및 텍스트를 마치 실제 존재하는 데이터처럼 만들었습니다.
7. defaultRecordGenerator는 공휴일과 같은 시스템 default 레코드를 읽기 전용으로 생성하는 프로그램이며, 이 프로그램이 생성하는 파일이 'public.dsv' 입니다. 로드시 디폴트로 읽게 되어 있으며, 존재하지 않으면 오류로 취급합니다. 현재는 어린이날만 있습니다.
8. D-day 설정 기능으로 인해 세이브파일이 호환되지 않습니다.
9. D-day 설정 기능으로 인해 Human-Readable 세이브 파일의 형식이 바뀌었습니다.
10. 랜덤 레코드 생성기를 업데이트 하여, 그럴듯한 제목과 텍스트를 생성하며, 샘플 개수도 20개로 늘었습니다.
11. `void printColorStrip(int colorNum)` 을 통해 0, 1, 2, 3(흰색 스트립, 녹색, 청색, 보라색)의 컬러 스티커를 출력 할 수 있도록 하였습니다.
12. 9999 시간에 대해 All Day Long 텍스트의 출력 및 최우선 정렬이 가능합니다.

13. (버그 수정) 공휴일로 고정된 일정이 계속해서 증식하는 문제를 해결했습니다.
14. Insert 함수 단에서 유효하지 않은 시간대를 걸러냅니다: 3을 리턴하면 유효하지 않은 날입니다.
15. Delete 관련 단에서 유효하지 않은 삭제를 걸러냅니다: 1을 리턴하면 존재하지 않는 값, 2를 리턴하면 시스템 고정(공휴일 등)을 삭제하려 시도한 경우입니다.
16. 14, 15를 UX단에서 popup으로 지원하도록 하였습니다.
17. System default 공휴일 관련 insert 문제를 해결했습니다.
18. 일일 북마크 종류에 따른 스티커가 표시됩니다.
19. 평일-토요일-일요일의 일자 색별 구분이 이루어집니다.
20. UX Layer에 리마인더 기능이 적용되었습니다.
21. getCommandScreen()이라는 범용 입력 함수가 제공됩니다.

<별첨>



```

(gdb) n
1143         if (ptr != NULL) length = (ptr - strbuf);
1: length = 0
2: strbuf = 0x7fffffffcc263 "consectetur adipiscing elit"
(gdb) n
1145         if (pos_SUR_end.col - col + 1 < length) {
1: length = 0
2: strbuf = 0x7fffffffcc263 "consectetur adipiscing elit"
(gdb) n
Breakpoint 1, print_date_ToDoWithDetails (targetDate=202405150000) at ploit_uxCore.c:1149
1149         mvprintw(row, col, "%s", length, strbuf);
1: length = 0
2: strbuf = 0x7fffffffcc263 "consectetur adipiscing elit"
(gdb) n
1150         col += length;
1: length = 0
2: strbuf = 0x7fffffffcc263 "consectetur adipiscing elit"
(gdb) n
1151         strbuf += length;
1: length = 0
2: strbuf = 0x7fffffffcc263 "consectetur adipiscing elit"
(gdb) n
1152         if (col > pos_SUR_end.col) {
1: length = 0
2: strbuf = 0x7fffffffcc263 "consectetur adipiscing elit"
(gdb) n
1156         if (ptr != NULL && length != 0){
1: length = 0
2: strbuf = 0x7fffffffcc263 "consectetur adipiscing elit"
(gdb) n
1141         while (*strbuf != '\0') {
1: length = 0
2: strbuf = 0x7fffffffcc263 "consectetur adipiscing elit"
(gdb) n
1142         ptr = strpbrk(strbuf, delimiters);
1: length = 0
2: strbuf = 0x7fffffffcc263 "consectetur adipiscing elit"
(gdb) n
1143         if (ptr != NULL) length = (ptr - strbuf);
1: length = 0
2: strbuf = 0x7fffffffcc263 "consectetur adipiscing elit"
(gdb) n
1145         if (pos_SUR_end.col - col + 1 < length) {
1: length = 0
2: strbuf = 0x7fffffffcc263 "consectetur adipiscing elit"
(gdb)

```

특정 조건 하에서 프로그램 실행이 멈추는 것을 확인했고, gdb 확인 결과 length = 0으로 고정된 상태에서 출력 대상 문자열이 아직 남은 상태로 무한 루프를 도는 것이 발견되었습니다.

보시다시피 공백을 기준으로 분리를 계속 시도하여 시작 인덱스가 0으로 고정 되는 것 같은데, 일단 임시방편으로, 코드 보시면 아시겠지만 문자열이 아직 남았고 length가 0이면 +1 해라 이런 식으로 고치긴 했는데, 위쪽에 문자열 토큰 기반 분리하는 부분을 한번 이번 주 모임에서 살펴봐야 할 것 같습니다.