일일 업무 사항 정리

작성자	제품팀 이민성 인턴
업무 일시	20221121~20221125

세부 사항

1. 업무 내역 요약 정리

목표 내역	Done & Plan
	9주차 수학 함수 - 절대값 - 나머지 - 난수 - 거듭제곱/제곱근 - 기타 수학 함수 탐색/정렬 함수 - 선형 탐색 - 이진 탐색 - 이진 트리 - 해시 테이블 관리 - 퀵 정렬 날짜/시간 - 시간 표시 - 형식 변환 - 기타 시간 관련

2. 내용 세부 (업무 세부 내역 정리 및 기타 사항 정리)

1. 수학 함수

- 절대값, 나머지, 거듭제곱/제곱근, 기타 수학 함수

헤더파일: math.h

함수	설명
삼각 함수	
double **sin** (double x);	사인 x 를 구한다.
double **cos** (double x);	코사인 x 를 구한다.
double **tan** (double x);	탄젠트 x 를 구한다.
역 삼각 함수	
double **asin** (double x);	아크 사인 x 를 구한다.
double **acos** (double x);	아크 코사인 x 를 구한다.
double **atan** (double x);	아크 탄젠트 x 를 구한다.
double **atan2** (double y, double x);	아크 탄젠트 y/x 를 구한다.
쌍곡선 함수	
double **sinh** (double x);	하이퍼볼릭 사인 x 를 구한다.
double **cosh** (double x);	하이퍼볼릭 코사인 x 를 구한다.
double **tanh** (double x);	하이퍼볼릭 탄젠트 x 를 구한다.
지수 · 대수 함수	
double **exp** (double x);	e^x 를 구한다.
double **frexp (double x, int * exp);**	지수를 exp 가 가리키는 변수에 저장하고 가수를 반환한다.
double **ldexp** (double x, int exp);	x * 2exp 를 반환한다.
double **log** (double x);	loge x 를 구한다.
double **log10** (double x);	log10x 를 구한다.
double **modf** (double x, double * intpart);	정수부를 intpart 가 가리키는 변수에 저장하고 소수부를 반환한다.
거듭제곱 · 거듭제곱근 · 올림 · 내림 · 절댓값 · 나머지 함수	
double **pow** (double x, double y);	x^y 를 구한다.
double **sqrt** (double x);	root(x)를 구한다.
double **ceil** (double x);	x 보다 작지 않은 가장 작은 정수를 구한다.
double **floor** (double x);	x 보다 크지 않은 가장 큰 정수를 구한다.
double **abs** (double x);	x 의 절댓값을 구한다.
double **fmod** (double x, double y);	x 를 y 로 나눈 나머지를 구한다.

```c
#include <stdio.h>
#include <math.h>
int main (void)
{
   printf ("sin(0) =  %lf₩n",sin(0));
   printf ("cos(0) =  %lf₩n",cos(0));
   printf ("tan(0) =  %lf₩n",tan(0));
   //단위는 라디안
   printf ("2의 8승 = %lf₩n", pow(2,8));
   printf ("루트 9= %lf₩n",  sqrt(9));
   printf ("-19의 절대값= %d₩n",  abs(-19));
   return 0;
```

- 난수

```c
#include <stdio.h>
#include <stdlib.h> //srand, rand를 사용하기 위한 헤더파일
#include <time.h> // time을 사용하기 위한 헤더파일

int main()
{
    srand(time(NULL)); // 난수 초기화
    for (int i = 0; i< 10; i++) // 10회 반복
    {
        int random = rand() % 5; // 0 ~ 4 사이의 숫자를 뽑아서 random 변수에 저장
        printf("%d ", random); // 출력
    }
    return 0;
}
```

```
2 1 3 0 2 2 4 2 3 4
```

먼저, rand함수는 rand % (숫자) 를 함으로써 원하는 숫자의 범위를 지정할 수 있다.

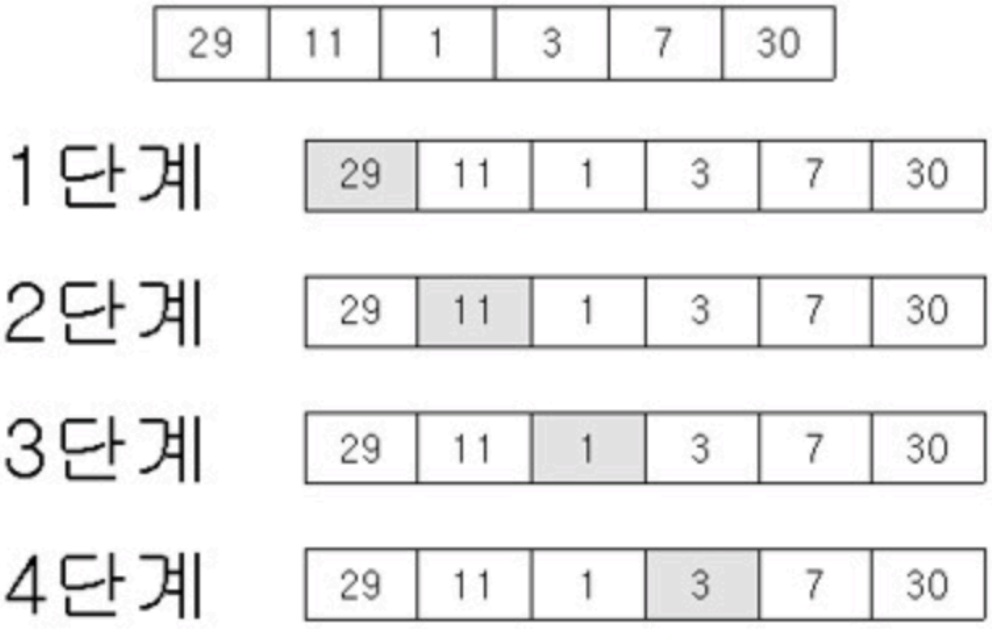
ex) rand() % 5 => 0 ~ 4 사이의 랜덤한 숫자

ex) (rand() % 5) + 1 => 1 ~ 5 사이의 랜덤한 숫자

그리고 7 번째 줄인 srand(time(NULL)); 이 필요한 이유는 컴퓨터 내부에 저장된 난수를 초기화 하기 위해서이다. srand(time(NULL)); 가 없게 되면 처음 실행할 때 생성된 난수가 저장이 되어서 다음 번 실행을 해도 계속 같은 결과값만 나오게 된다.

2. 탐색/정렬 함수

- 선형 탐색 함수



선형 탐색이란 원하는 레코드를 찾을 때까지 레코드를 처음부터 끝까지 차례로 하나씩 비교하면서 검색하는 것이다. 단순한 방식으로 정렬되지 않는 검색에 가장 유용하며 평균 검색시간이 많이 걸 리는 단점이 있다. 순차 검색 (sequential searching), 순서 검색, Linear Search, Sequential Search 라고도 한다.

lsearch

기능 : 테이블에서 데이터를 선형 탐색 한다. 찾지 못하면 테이블에 추가

기본형 : void *lsearch(const void *key, void *base, size_t *nmemb, size_t size, int (*compare)(const void *));

key : 찾고자 하는 데이터

base : 테이블의 첫 번째 데이터

nmemb : 데이터의 개수

size : 데이터의 크기

compare : 두 데이터를 비교하는 함수로 같으면 0, 다르면 0 이 아닌 값 반환

반환값 : 찾은 데이터에 대한 포인터를 반환. 만약 발견하지 못하면 테이블 끝에 추가, 추가한 데이터 포인터 반환

헤더파일 : search.h

lfind

기능 : 테이블에서 데이터를 선형 탐색 한다. 찾지 못하면 테이블에 추가

기본형 : void *lfind(const void *key, void *base, size_t *nmemb, size_t size, int (*compar)(const void *));

key : 찾고자 하는 데이터

base : 테이블의 첫 번째 데이터

nmemb : 데이터의 개수

size : 데이터의 크기

compare : 두 데이터를 비교하는 함수로 같으면 0, 다르면 0 이 아닌 값 반환

반환값 : 찾은 데이터에 대한 포인터를 반환. 만약 발견하지 못하면 NULL반환

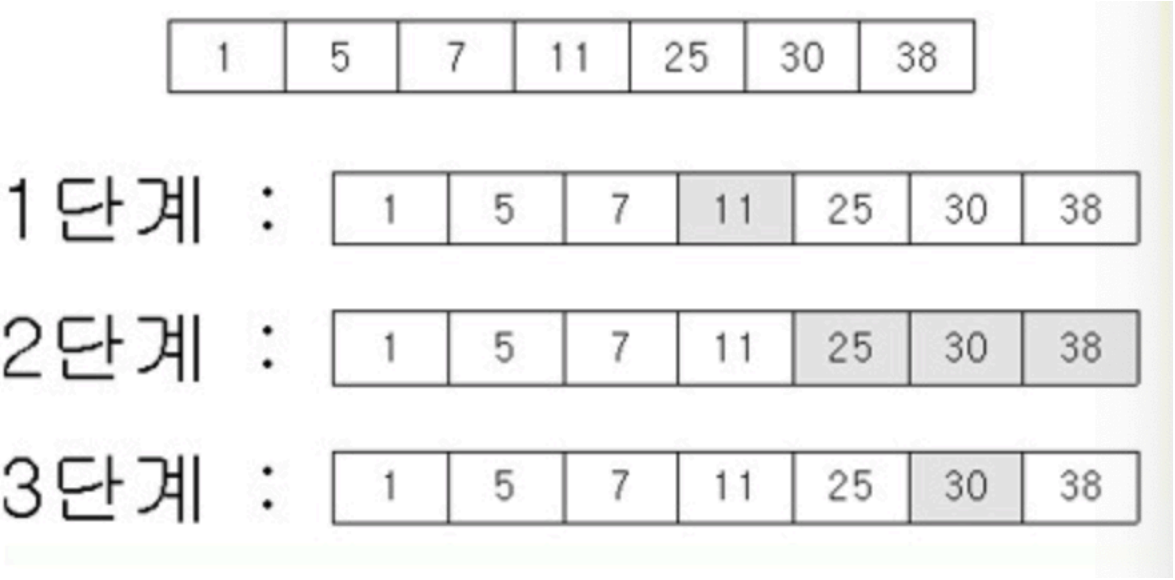
헤더파일 : search.h

```
#include <stdio.h>
#include <search.h> // lsearch 함수가 정의된 헤더 파일
#include <string.h> // strcmp 함수가 정의된 헤더 파일
#define TABLESIZE 10 // 테이블의 크기
#define ELEMENTSIZE 15 // 데이터의 크기

int compare(const void *a, const void *b);
main()
{
        char table[TABLESIZE][ELEMENTSIZE]={"Boan", "Project", "Forever"};
        char *ptr;
        int datanum=3; // 데이터 개수
        // table에서 Linux를 탐색하고 찾은 데이터에 대한 포인터 반환
        ptr = (char *)lsearch("Project", table, &datanum, ELEMENTSIZE, compare);
        printf("%s\n", ptr);
        // table에서 "Programming"을 찾는데 없으면 테이블 뒤에 추가하고 이에
        // 대한 포인터를 반환. 테이블에 데이터가 하나 추가되므로 datanum 1증가
ptr = (char *)lsearch("Fithing!", table, &datanum, ELEMENTSIZE, compare);
        printf("%s\n", ptr);
}

// a와 b를 비교해 같으면 0을 반환
int compare(const void *a, const void *b)
{
        return strcmp((char *)a, (char *)b);
}
~
```

- 이진 탐색 함수



일정한 순서로 배열된 순서 파일에서 중간 레코드 키(Kn)와 R를 비교해 결과에 의해 찾아가는 검색 방법이다. 레코드들이 키 값에 따라서 정렬되어 있어야 한다. 많은 자료에서 검색할 때 효율적이다.

최대 비교 횟수 : log2n + 1

bsearch

기능 : 테이블에서 데이터를 이진 탐색한다. 발견하지 못하면 NULL반환

기본형 : void *bsearch(const void *key, void *base, size_t *nmemb, size_t size, int (*compare)(const void *));

key : 찾고자 하는 데이터

base : 테이블의 첫 번째 데이터

nmemb : 데이터의 개수

size : 데이터의 크기

compare : 두 데이터를 비교하는 함수로 같으면 0, 다르면 0 이 아닌 값 반환

반환값 : 찾은 데이터에 대한 포인터를 반환. 만약 발견하지 못하면 NULL반환

헤더파일 : stdlib.h

```c
#include <stdio.h>
#include <stdlib.h>    /* bsearch 함수가 정의된 헤더 파일 */
#include <string.h>    /* strcmp 함수가 정의된 헤더 파일 */
#define TABLESIZE 5    /* 테이블의 크기 */
#define ELEMENTSIZE 10    /* 데이터의 크기 */

int compare(const void *a, const void *b);
main()
{
    /* table은 정렬되어 있어야 함 */
    char table[TABLESIZE][ELEMENTSIZE]={"A", "B", "C", "D", "E"};
    char *ptr;

    if((ptr=(char *)bsearch("B", table, TABLESIZE, ELEMENTSIZE, compare))==
ULL)
        printf("Not found\n");
    else
        printf("%s\n", ptr);

    if((ptr=(char *)bsearch("Z", table, TABLESIZE, ELEMENTSIZE, compare))==
ULL)
        printf("Not found\n");
    else
        printf("%s\n", ptr);
}
int compare(const void *a, const void *b)
{
    return strcmp((char *)a, (char *)b);
}
```

```
bitnang@ubuntu:~/system/source/13장/13_1$ ./a.out
B
```

스크린샷 27,1 Al

- 이진 트리

트리(일반 트리) 중에서 자식 노드의 수가 2 개 이하인 것을 이진 트리(binary tree)라고 한다.

일반 트리는 앞에서 보았듯이 컴퓨터에 표현하기 어렵기 때문에 이진 트리로 바꾼다.

tsearch

기능 : 이진 트리에서 데이터를 탐색, 발견하지 못하면 트리에 추가

기본형 : void *tsearch(const void *key, void **rootp, int (*compar)(const void *, const void *));

key : 찾고자 하는 데이터

rootp : 이진 트리에 대한 포인터

compare : 두 데이터를 비교하는 함수로 같으면 0, 다르면 0 이 아닌 값 반환

반환값 : 찾은 데이터에 대한 포인터 반환, 만약 발견하지 못하면 트리에 추가하고 추가한 데이터에 대한

포인터 반환.
헤더파일 : search.h

tfind

기능 : 이진 트리에서 데이터를 탐색. 발견하지 못하면 NULL반환
기본형 : void *tfind(const void *key, void **rootp, int (*compar)(const void *, const void *));
key : 찾고자 하는 데이터
rootp : 이진 트리에 대한 포인터
compare : 두 데이터를 비교하는 함수로 같으면 0, 다르면 0 이 아닌 값 반환
반환값 : 찾은 데이터에 대한 포인터 반환, 만약 발견하지 못하면 NULL 반환.
헤더파일 : search.h

tdelete

기능 : 이진 트리에서 데이터삭제
기본형 : void *tdelete(const void *key, void **rootp, int (*compar)(const void *, const void *));
key : 삭제 하고자 하는 데이터
rootp : 이진 트리에 대한 포인터
compare : 두 데이터를 비교하는 함수로 같으면 0, 다르면 0 이 아닌 값 반환
반환값 : 삭제한 데이터에 대한 부모 노드 포인터 반환, 만약 삭제 하려는 데이터가 없으면 NULL을 반환
헤더파일 : search.h

twalk

기능 : 이진 트리를 방문한다.
기본형 : void twalk(const void *root, void (*action)
(const void *nodep, const VISIT which, const int depth));
root : 이진 트리에 대한 포인터
action : 구체적인 동작을 하는 함수
반환값 : 없음
헤더파일 : search.h

- 해시 테이블 관리

해시는 자료 입력할 때 검색하기 쉬운 위치에 삽입하는 방법이다.
검색 방법이라기 보단 빠른 검색을 위한 자료 관리 알고리즘이다.
검색 알고리즘 중 빠른 반면 메모리소모가 심하다.
충분한 메모리 확보 필요, 메모리가 넉넉하지 못하면 잦은 충돌이 발생한다.

hsearch

기능 : 해시 테이블에서 데이터를 탐색
기본형 : ENTRY *hsearch(ENTRY item, ACTION action);
item : 찾고자 하는 데이터

action : FIND = 발견하지 못하면 NULL반환

ENTER = 발견하지 못하면 삽입하고 삽입한 데이터 포인터 반환

반환값 : 찾은 데이터에 대한 포인터 반환

헤더파일 : search.h

hdestory

기능 : 해시 테이블을 제거

기본형 : void hdestory(void);

반환값 : 없음

헤더파일 : search.h

```
#include <stdio.h>
#include <search.h>   /* hcreate, hsearch 함수가 정의된 헤더 파일 */

main()
{
   ENTRY item;
   ENTRY *result;



            // 해시 테이블 생성. 5는 테이블에 저장할 데이터의 개수에 대한 추정값
   hcreate(5);



            // 3개의 데이터를 해시 테이블에 삽입
   item.key = "Boan";
   item.data = "Boan";
            // item을 해시 테이블에서 탐색하는데, 발견하지 못하면 item을 삽입하고  이에 대한 포인터를 반환
   hsearch(item, ENTER);
   item.key = "Project";
   item.data = "Project";
   hsearch(item, ENTER);
   item.key = "Forever";
   item.data = "Forever";
   hsearch(item, ENTER);



   item.key = "Project";
// 해시 테이블에서 데이터를 탐색하여 item을 발견하면 데이터를 출력하고 발견하지 못하면 오류출력

   if((result=hsearch(item, FIND)) == NULL)
      printf("Not found\n");
   else
      printf("Found : %s\n", result->data);



   item.key = "ZZANG";
   if((result=hsearch(item, FIND)) == NULL)
      printf("Not found\n");
   else
      printf("Found : %s\n", result->data);
}
```

```
Found : Project
Not found
```

- 퀵 정렬 함수

이름 그대로 상당히 빠른 정렬의 알고리즘이다

기준값을 정하고 전방과 후방의 값들을 기준값과 비교하여 전방값이 기준값보다 큰가를 체크, 후방값이 기준값보다 작은가를 체크한다.

한번 정렬이 일어 났으면 그 배열을 반으로 나누어 위와 같이 반복한다.

qsort
기 능 : 퀵 정렬을 한다.
기본형 : void qsort(void *base, size_t nmemb, size_t size, int(*compare)(const void *, const void *));
base : 정렬할 테이블의 첫 번째 데이터
nmemb : 데이터의 수
size : 데이터의 크기
compare : 두 데이터의 크기를 비교하는 함수로 첫 번째 데이터가 작으면 음수, 같으면 0, 크면 양수 반환
반환값 : 없음
헤더파일 : stdlib.h

3. 날짜/시간

1. time 헤더 파일 포함

```
#include <time.h> //C 언어
```

2. time() 함수를 호출하여 현재의 날짜, 시간을 얻어 time_t 변수에 저장

```
time_t timer = time(NULL);
```

time 함수는 time_t 결과값으로 타입을 리턴한다. 결과값은 Unix 운영체제가 공식 출시한 1970 년 1 월 1 일 0 시 0 분 0 초를 기점으로 현재까지 흐른 시간을 초단위로 나타낸다.

3. localtime() 함수를 호출하여 포맷 변환하기

```
struct tm* t = localtime(&timer);
```

time 함수가 리턴해주는 값은 시스템에 따라 32 비트나 64 비트 정수이다. 이를 그대로 사용하기에는 무리가 있다. 따라서 사용하고자 하는 형태에 맞게끔 포맷팅을 해주어야 한다. 이 과정을 localtime 함수를 사용하여 tm 구조체를 활용, 원하는 형태로 변환한다.

struct tm 구조체

```
struct tm {
   int tm_sec;         // 초,  range 0 to 59
   int tm_min;         // 분, range 0 to 59
   int tm_hour;        // 시간, range 0 to 23
   int tm_mday;        // 일, range 1 to 31
   int tm_mon;         // 월, range 0 to 11
   int tm_year;        // 1900 년 부터의 년
   int tm_wday;        // 요일, range 일(0) to 토(6)
   int tm_yday;        // 1 년 중 경과 일, range 0 to 365
   int tm_isdst;       // 섬머타임 실시 여부 (양수, 0, 음수)
};
```

시간 구조체는 위와 같이 정의되어 있다. 여기서 주의할 점은 tm_year과 tm_mon의 사용법입니다. tm_year은 1900 년도부터의 년이기에 출력하고 싶은 년도를 1900 년도에 +해서 사용해야 하고 tm_mon은 1 월부터 12 월까지 출력하기 위해서는 +1 을 해서 사용하여야 한다.

시간 출력 예제

```
#define _CRT_SECURE_NO_WARNINGS // 혹은 localtime_s 를 사용
#include <stdio.h>
#include <time.h>

int main() {
    time_t timer;
    struct tm* t;
    timer = time(NULL); // 1970 년 1 월 1 일 0 시 0 분 0 초부터 시작하여 현재까지의 초
    t = localtime(&timer); // 포맷팅을 위해 구조체에 넣기

    printf("유닉스 타임 (Unix Time): %lld 초\n\n", timer);
    printf("현재 년: %d\n", t->tm_year + 1900);
    printf("현재 월: %d\n", t->tm_mon + 1);
    printf("현재 일: %d\n", t->tm_mday);
    printf("현재 시: %d\n", t->tm_hour);
    printf("현재 분: %d\n", t->tm_min);
    printf("현재 초: %d\n", t->tm_sec);
    printf("현재 요일: %d\n", t->tm_wday); // 일=0, 월=1, 화=2, 수=3, 목=4, 금=5, 토=6
    printf("올해 몇 번째 날: %d\n", t->tm_yday); // 1 월 1 일은 0, 1 월 2 일은 1
    printf("서머타임 적용 여부: %d\n", t->tm_isdst); // 실시 중이면 양수, 미실시면 0, 실시 정보가
없으면 음수
```

```
    return 0;
}
```

```
유닉스 타임 (Unix Time): 1611420167 초

현재 년: 2021
현재 월: 1
현재 일: 24
현재 시: 1
현재 분: 42
현재 초: 47
현재 요일: 0
올해 몇 번째 날: 23
서머타임 적용 여부: 0
```