# 자료구조 (Data Structure)
# 7주차: 정렬

# 지난 시간 요약

- heap sort: 시간 $O(n \log n)$,
  공간 $O(1)$
- quick sort: 시간 평균 $O(n \log n)$ 최악 $O(n^2)$,
  공간 $O(\log n)$
- insertion sort: 시간 최선 $O(n)$, 최악 $O(n \log n)$,
  공간 $O(n)$

# 오늘의 목차

- 배열에 저장된 트리 출력하기 (C 코드)
- heap sort (리눅스 커널의 최초 버전)

# 트리 출력 - printf

```
#include <stdio.h>
int main()
{
        printf("Hello");
        return 0;
}
```

# 트리 출력 - printf

int printf(const char *format, ...);

- char *는 글자 값이 저장된 주소
- const char *는 해당 변수를 통해서 글자값을 바꾸지 않겠다는 약속

# 트리 출력 - printf - 글자 배열

```
#include <stdio.h>
int main()
{
	char arr[5];
	arr[0] = 'H';
	arr[1] = 'i';
	arr[2] = '\0';
	printf(arr);
	return 0;
}
```

# 트리 출력 - printf - 글자 배열

- 글자에 해당하는 값은 작은 따옴표로 표시한다.
- 글자 배열을 문장으로 다룰 때, 마지막 글자는 '\0'라는 특수문자로 쓴다.

# 트리 출력 - printf - 글자 배열에 초기값

```
#include <stdio.h>
int main()
{
	char arr[5] = {'H', 'i', '\0'};
	printf(arr);
	return 0;
}
```

# 트리 출력 - printf - 글자 배열에 초기값

```
#include <stdio.h>
int main()
{
	char arr[5] = "Hi";
	printf(arr);
	return 0;
}
```

# 트리 출력 - printf - 글자 배열에 초기값

- 배열을 처음 만들 때, 초기값들을 중괄호 { } 안에 적어 줄 수 있다.
- 문자 배열의 경우, 초기값들을 큰 따옴표"" 안에 적어 줄 수 있다.
- 공간이 충분하면 마지막에 '\0'도 붙는다.

# 트리 출력 - printf - 특수문자 '\0'

```
#include <stdio.h>
int main()
{
	char arr[5] = {"H\0i"};
	printf(arr);
	return 0;
}
```

# 트리 출력 - printf - 특수문자 '\0'

- 글자 배열에서 글자를 하나씩 처리하다가 '\0'을 발견하면 문장이 끝났다고 보고 처리를 종료한다.

# 트리 출력 - printf - 포맷문자열 %c, %d

```
#include <stdio.h>
int main()
{
	char arr[5] = {"H\0i"};
	for (int i = 0; i < 4; i++)
		printf("%c[%d]\n", arr[i], arr[i]);
	return 0;
}
```

# 트리 출력 - printf - 포맷문자열 %c, %d

- %로 표시된 자리에는 다음 매개변수의 값을 적절한 서식에 맞춰 대입한다.
- %c는 문자 값을 문자로 나타낸다.
- %d는 숫자 값을 숫자로 나타낸다.
- \n은 줄바꿈 특수문자를 의미한다.

# 트리 출력 - printf - 포맷문자열의 서식 너비

```
#include <stdio.h>
int main()
{
        printf("%5c\n", 'A');
        printf("%4c\n", 'B');
        printf("%3c\n", 'C');
        printf("%2c\n", 'D');
        printf("%1c\n", 'E');
        return 0;
}
```

# 트리 출력 - printf - 포맷문자열의 서식 너비

- %d와 서식 문자 사이에 숫자를 적으면, 해당 항목의 최소 너비를 정할 수 있다.

# 출력 - printf - 포맷문자열의 가변 서식 너비

```
#include <stdio.h>
int main()
{
	for (int i = 1; i <= 5; i++)
		printf("%*c\n", 'z');
	return 0;
}
```

# 출력 - printf - 포맷문자열의 가변 서식 너비

- %d와 서식 문자 사이에 *을 적으면, 해당 항목의 최소 너비를 매개변수로 전달할 수 있다.

# 트리 출력 - 계획

----------32----------

-------16-------A------16-------

---8----B------16-------C---8---

-4-D----8---E---8---F---8---G-4-

# 트리 출력 - 계획 1 / 2

```
#include <stdio.h>
int main()
{
        printf("%16c\n", 'A');

        printf("%8c", 'B');
        printf("%16c\n", 'C');
```

# 트리 출력 - 계획 2 / 2

```
        printf("%4c", 'D');
        printf("%8c", 'E');
        printf("%8c", 'F');
        printf("%8c\n", 'G');

        return 0;
}
```

# 트리 출력 - 층별 출력 계획 1 / 3

```
#include <stdio.h>
int main()
{
        int tree_width = 32;
        char data[10] = "ABCDEFG";
        int level = 1;
        int level_size = 1;
        int level_width = tree_width;
        int level_start = 0;
        int level_end = 0;
```

# 트리 출력 - 층별 출력 계획 2 / 3

```
for (int i = 0; i < 7; i++) {
        int spaces = level_width;
        if (i == level_start)
                spaces /= 2;
        printf("%*c", spaces, data[i]);
```

# 트리 출력 - 층별 출력 계획 3 / 3

```
            if (i == level_end) {
                printf("\n");
                level += 1;
                level_size *= 2
                level_width = tree_width / level_size;
                level_start = level_end + 1;
                level_end += level_size;
            }
        }
}
```

# 트리 출력 - print_tree() 1 / 3

```
void print_tree(char *data, int tree_size, int tree_width)
{
        int level_size = 1;
        int node_width = tree_width;
        int level_start = 0;
        int level_end = 0;

        for (int i = 0; i < tree_size; i++) {
```

# 트리 출력 - print_tree() 2 / 3

```
int spaces = node_width;

if (i == level_start)
        spaces /= 2;

printf("%*c", spaces, data[i]);
```

# 트리 출력 - print_tree() 3 / 3

```
            if (i == level_end) {
                printf("\n");
                level_size *= 2;
                node_width = tree_width / level_size;
                level_start = level_end + 1;
                level_end += level_size;
            }
        }
        printf("\n");
}
```

# 트리 출력 - print_tree() 테스트

```
int main()
{
        screen_width = 50;
        char data[10] = "ABCDEFG";
        print_tree(data, 7, screen_width);
        return 0;
}
```

# heap sort - 최대힙 만들기 - swap 함수

```
void swap(char *x, char *y)
{
        char t = *x;
        *x = *y;
        *y = t;
}
```

# heap sort - 최대힙 만들기 - swap 함수

```
int main()
{
        char data[10] = "ABCDEFG";
        swap(data + 5, data + 6);
        print_tree(data, 7, 32);
}
```

# heap sort - 최대힙 만들기 - parent 함수

```
int parent(int child)
{
        return (child - 1) / 2;
}
```

# heap sort - 최대힙 만들기 - parent 함수

```
int main()
{
        char data[10] = "ABCDEFG";
        swap(data + parent(6), data + 6);
        swap(data + parent(3), data + 3);
        print_tree(data, 7, 32);
}
```

# heap sort - 최대힙 만들기 - left_child 함수

```
int left_child(int parent)
{
        return 2 * parent + 1;
}
```

# heap sort - 최대힙 만들기 - left_child 함수

```
int main()
{
        char data[10] = "ABCDEFG";
        swap(data + left_child(2), data + 2);
        swap(data + left_child(1), data + 1);
        print_tree(data, 7, 32);
}
```

# heap sort - 최대힙 만들기 - 최대 자식 찾기

```
int main() {
        char data[10] = "ABCDEFG";
        int r = parent(6);
        int c = left_child(r);
        if (data[c] < data[c + 1])
                c = c + 1;
        if (data[r] < data[c])
                swap(data + r, data + c);
        print_tree(data, 7, 32);
}
```

# heap sort - 최대힙 만들기 - level 2 이하 1/2

```
int main()
{
        char data[10] = "ABCDEFG";
        int r = 2;
        int c = left_child(r);
        if (data[c] < data[c + 1])
                c = c + 1;
        if (data[r] < data[c])
                swap(data + r, data + c);
```

# heap sort - 최대힙 만들기 - level 2 이하 2/2

```
        r = 1;
        c = left_child(r);
        if (data[c] < data[c + 1])
                c = c + 1;
        if (data[r] < data[c])
                swap(data + r, data + c);
        print_tree(data, 7, 32);
}
```

# heap sort - 최대힙 만들기 - level2 짧게 1/2

```
int main()
{
        char data[10] = "ABCDEFG";
        int i = 2, r, c;
        for ( ; i >= 1; i -= 1) {
                r = i;
                c = left_child(r);
```

# heap sort - 최대힙 만들기 - level2 짧게 2/2

```
            if (data[c] < data[c + 1])
                c = c + 1;
            if (data[r] < data[c])
                swap(data + r, data + c);
        }
    }
```

# heap sort - 최대힙 만들기 - level 1 1/3

```
int main()
{
        char data[10] = "ABCDEFG";
        int i = 2, r, c;

        for ( ; i >= 1; i -= 1) {
                r = i;
                c = left_child(r);
                if (data[c] < data[c + 1])
                        c = c + 1;
```

# heap sort - 최대힙 만들기 - level 1 2/3

```
        if (data[r] < data[c])
                swap(data + r, data + c);
}

r = 0;
c = left_child(r);
if (data[c] < data[c + 1])
        c = c + 1;
if (data[r] < data[c])
        swap(data + r, data + c);
```

# heap sort - 최대힙 만들기 - level 1 3/3

```
        r = c;  c = left_child(r);
        if (data[c] < data[c + 1])
                c = c + 1;
        if (data[r] < data[c])
                swap(data + r, data + c);
        print_tree(data, 7, 32);
}
```

# heap sort - 최대힙 만들기 - level 1 짧게 1/2

```
int main()
{
        char data[10] = "ABCDEFG";
        int i = 2, r, c;
        for ( ; i >= 0; i -= 1) {
                for (r = i; left_child(r) < 7; r = c) {
                        c = left_child(r);
```

# heap sort - 최대힙 만들기 - level 1 짧게 2/2

```
                if (data[c] < data[c + 1])
                    c = c + 1;
                if (data[r] >= data[c])
                    break;
                swap(data + r, data + c);
            }
        }
        print_tree(data, 7, 32);
}
```

# heap sort - 최대힙 만들기 - right_child 존재

```
int main()
{
        char data[10] = "ABCDEFG";
        int i = 2, r, c;

        for ( ; i >= 0; i -= 1) {
                for (r = i; left_child(r) < 7; r = c) {
                        c = left_child(r);
```

# heap sort - 최대힙 만들기 - right_child 존재

```
                if (c + 1 < 7 && data[c] < data[c + 1])
                        c = c + 1;

                if (data[r] >= data[c])
                        break;
                swap(data + r, data + c);
                }
        }
        print_tree(data, 7, 32);
}
```

# heap sort - 최대힙 만들기 1/2

```
int main()
{
        char data[10] = "ABCDEFG";
        int n = 7;

        int i = parent(n - 1), c, r;
        for ( ; i >= 0; i -= 1) {
                for (r = i; r * 2 < n; r = c) {
                        c = left_child(r);
```

# heap sort - 최대힙 만들기 2/2

```
                if (c < n - 1 && data[c] < data[c + 1])
                        child += 1;
                if (data[r] >= data[c])
                        break;
                swap(data + r, data + c);
        }
    }
    print_tree(data, 7, 32);
}
```

# heap sort - 정렬하기 - G 처리 1/3

```
int main()
{
        char data[10] = "GEFDBAC";
        int n = 7;

        swap(data, data + 6);
        n = 6;
        int r = 0;
        int c = left_child(r);
```

# heap sort - 정렬하기 - G 처리 2/3

```
if (c < n - 1 && data[c] < data[c + 1])
        c += 1;
if (data[r] < data[c])
        swap(data + r, data + c);

r = c;
c = left_child(r);
```

# heap sort - 정렬하기 - G 처리 3/3

```
        if (c < n - 1 && data[c] < data[c + 1])
                child += 1;
        if (data[r] < data[c])
                swap(data + r, data + c);
        print_tree(data, 7, 32);
}
```

# heap sort - 정렬하기 - G 처리 짧게 1/2

```
int main()
{
        char data[10] = "GEFDBAC";
        int n = 7;

        swap(data, data + 6);
        n = 6;

        int c, r;
```

# heap sort - 정렬하기 - G 처리 짧게 2/2

```
        for (r = 0; left_child(r) < n; r = c) {
                c = left_child(r);
                if (c < n - 1 && data[c] < data[c + 1])
                        child += 1;
                if (data[r] >= data[c])
                        break;
                swap(data + r, data + c);
        }
        print_tree(data, 7, 32);
}
```

# heap sort - 정렬하기 - F 처리 1/3

```
int main()
{
        char data[10] = "GEFDBAC";
        int n = 7;

        swap(data, data + 6);
        n = 6;
        int c, r;
```

# heap sort - 정렬하기 - F 처리 2/3

```
for (r = 0; left_child(r) < n; r = c) {
        c = left_child(r);
        if (c < n - 1 && data[c] < data[c + 1])
                child += 1;
        if (data[r] >= data[c])
                break;
        swap(data + r, data + c);
}
swap(data, data + 5);
n = 5;
```

# heap sort - 정렬하기 - F 처리 3/3

```
    for (r = 0; left_child(r) < n; r = c) {
            c = left_child(r);
            if (c < n - 1 && data[c] < data[c + 1])
                    child += 1;
            if (data[r] >= data[c])
                    break;
            swap(data + r, data + c);
    }
    print_tree(data, 7, 32);
}
```

# heap sort - 정렬하기 - F 처리 짧게 1/2

```
int main()
{
        char data[10] = "GEFDBAC";
        int n = 7;

        int i, c, r;
        for (i = n - 1; i >= 0; i -= 1) {
                swap(data, data + i);
                for (r = 0; left_child(r) < i; r = c) {
```

# heap sort - 정렬하기 - F 처리 짧게 2/2

```
                c = left_child(r);
                if (c < i - 1 && data[c] < data[c + 1])
                        c = c + 1;
                if (data[r] >= data[c])
                        break;
                swap(data + r, data + c);
        }
    }
    print_tree(data, 7, 32);
}
```

# heap sort - 정렬 함수 1 / 3

```
void sort(char *data, int n)
{
        int i = parent(n - 1), c, r;
        /* heapify */
        for ( ; i >= 0; i -= 1) {
                for (r = i; left_child(r) < n; r = c) {
                        c = left_child(r);
                        if (c < n - 1 && data[c] < data[c + 1])
                                c += 1;
```

# heap sort - 정렬 함수 2 / 3

```
                if (data[r] >= data[c])
                        break;
                swap(data + r, data + c);
        }
}

/* sort */
for (i = n - 1; i >= 0; i -= 1) {
        swap(data, data + i);
```

# heap sort - 정렬 함수 3 / 3

```
            for (r = 0; left_child(r) < i; r = c) {
                c = left_child(r);
                if (c < i - 1 && data[c] < data[c + 1])
                    c += 1;
                if (data[r] >= data[c])
                    break;
                swap(data + r, data + c);
            }
        }
}
```

# heap sort - 정렬 함수 테스트

```
int main()
{
	char data[10] = "GFEDCBA";
	int n = 7;

	sort(data, n);

	print_tree(data, 7, 32);
}
```

# 요약

- heap sort를 C로 구현

# COUNTING_SORT

- 데이터의 개수를 세어 정렬하는 방법

# COUNTING_SORT 과정

1. 데이터 개수를 세는 배열 생성 및 초기화

2. 각 원소의 개수 세기

3. 출력 배열에 정렬된 원소 배치

# COUNTING_SORT 시각화

- 예시: [5, 5, 2, 2, 1] 정렬 과정
- 최대값: 5
- 카운트 배열: [0, 1, 2, 0, 0, 2]
- 최종 정렬 결과: [1, 2, 2, 5, 5]

# COUNTING_SORT

함수 COUNTING_SORT(리스트 L, 정수 max_value):

- count = 새로운 배열(max_value + 1) // 0으로 초기화
- 반복: value는 L의 모든 데이터

        count[value] += 1

- sorted = 새로운 리스트(L->data, 0)
- 반복: value는 0부터 max_value까지

        반복: count[value]번

                ALIST_PUSH(sorted, value)

- 삭제: count, sorted

# COUNTING_SORT의 시간 복잡도

개수 배열 초기화의 시간복잡도:

- O(k)

데이터 개수 세기:

- O(n)

총 O(n + k)