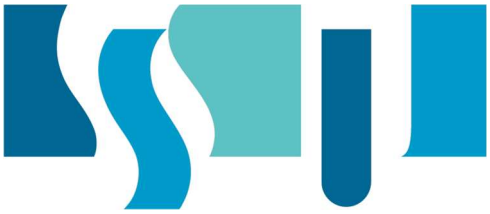

HW#02 Programming Assignment

- Bubble / Insertion Sorting -

과목명	컴퓨터수학 I
교수명	최 종 선
학 과	컴퓨터학부
학 번	20222904
이 름	정 해 성
제출일	2022.05.19

목차

I. 서론

버블 정렬과 삽입 정렬을 응용하여 문자열이 들어있는 배열을 오름차순, 또는 내림차순으로 정렬하고, C언어 코드로 구현하는 것이 이번 레포트의 목표이다.

본론의 1장에서는 먼저, 문자열을 사전순으로 비교하는 원리와 이를 구현한 함수를 소개한다. 그리고 2장과 3장에는 버블 정렬과 삽입 정렬의 대략적인 알고리즘을 기술하고 전체 C언어 코드를 보여주는 구조로 구성 되어있다.

II. 본론

1. 문자열 비교 원리 및 함수 설명

정렬을 하기 위해서는 각 아이템의 크고 작음, 또는 우선순위를 비교할 수 있어야 한다. 하지만 숫자와 달리 프로그래밍에서 문자열의 우선순위를 비교하는 것은 단순하지 않다. 따라서 정렬 알고리즘을 구현하기 전에 두 문자열의 우선순위를 비교하는 함수를 구현했다.

문자열 "track"은 "tree"보다 사전순으로 앞서 있다. 우리는 이 사실을 첫 번째 문자부터 차례대로 비교하며 간단히 확인할 수 있다. 처음으로 다른 문자가 나왔을 때(위의 예시에서는 'a'와 'e'), 이 두 문자를 알파벳 순서로 비교하는 것이다. 아래의 코드는 방금 우리가 머릿속에서 실행한 알고리즘을 함수로 구현한 것이다.

```
int compareString(char* a, char* b) {

	while (*a != '\0' && *b != '\0' && *a == *b) a++, b++;

	if (*a == *b) return 0;
	if (*b == '\0' || *a > *b) return 1;
	if (*a == '\0' || *a < *b) return -1;
	return 0;
}
```

compareString 함수는 문자열 두 개를 매개변수로 입력 받아 a 가 b 와 같으면 0, b 보다 크면 1, 작으면 -1 을 반환한다. 여기서 문자열의 크거나 작다는 의미는 사전순으로 우선순위가 뒤쳐져 있거나 앞서 있다는 뜻이다.

내부 코드의 첫째 줄은 while 문으로 두 문자열이 다를 때까지 다음 문자로 이동하는 코드이다. 또한 문자열의 맨 마지막은 널 문자('₩0')로 끝난다는 사실을 이용하여 두 문자열 중 하나라도 마지막에 도달하면 종료된다. while 문을 끝내고 나서 아래의 if 문으로 조건을 검사하여 0, 1, -1 중 하나를 반환하는데, 먼저 두 문자가 같은 조건을 검사한다. 앞의 while 문을 거쳤음에도 불구하고 두 문자가 같다는 뜻은 서로 똑같이 마지막에 도달했다는 뜻이고, 즉 두 문자열이 완전히 같다는 것이므로 0 을 반환한다.

사전순으로 비교하는 기준에는 또 다른 한 가지가 있다. 그것은 문자열의 길이가 더 짧은 것이 사전순으로 앞서 있다는 것이다. 예를 들어 "green"은 "greenhouse"보다 앞서 있다. 이것을 조건문으로 확인하기 위해서는 "green"과 "greenhouse" 중 무엇이 먼저 마지막에 도달했는지를 검사해야 한다. 두 번째와 세 번째 조건문에서는 이러한 경우를 검사하는 알고리즘이 포함되어 있다.

이제 이 함수를 사용하여 반환 값이 0 보다 큰지, 혹은 작은지를 검사하여 두 문자열의 우선순위를 비교할 수 있다.

2. Bubble Sorting

'거품 정렬'이라고도 불리는 Bubble Sorting 은 인접한 두 아이템을 비교한 후 서로 자리를 바꾸는 아이디어에서 시작한다. 구체적으로 설명하면 i 번째 아이템이 i+1 번째 아이템보다 크다면 자리를 바꾸는 것이고, i 는 리스트의 끝에 다다를 때까지 증가한다. 문자열을 Bubble Sorting 으로 정렬하기 위해서 앞서 만들어 두었던 compareString 함수를 활용할 것이다. 오름차순으로 정렬할 때 이 함수의 반환 값이 0 보다 크면 자리를 바꾼다.

```
// 버블 소팅 방법으로 리스트를 오름차순으로 정렬하는 함수
void bubbleSort_ASC(char* arr[], int size) {

	for (int i = 1; i < size; i++) {
		for (int j = 0; j < size - i; j++) {

			// arr[j+1]가 arr[j]보다 사전 순으로 앞서 있을 때
			if (compareString(arr[j], arr[j + 1]) > 0) {
				// arr[j]와 arr[j+1] 스왑
				char* tmp;
				tmp = arr[j];
				arr[j] = arr[j + 1];
				arr[j + 1] = tmp;
			}
		}
	}
}
```

bubbleSort_ASC 함수는 매개변수로 받은 문자열 배열 arr 을 Bubble Sorting 을 사용하여 오름차순으로 정렬한다. 내림차순으로 정렬할 때는 부등호의 방향만 바꾸어 주면 된다.

i. 전체 C 언어 구현 코드

```c
#include <stdio.h>
#define SIZE 18

/* 사전순을 기준으로 문자열을 비교하여
* str1 > str2 일 때 1,
* str1 = str2 일 때 0,
* str1 < str2 일 때 -1을
* 반환하는 함수
*/
int compareString(char* a, char* b) {

	while (*a != '\0' && *b != '\0' && *a == *b) a++, b++;

	if (*a == *b) return 0;	// *a와 *b가 같다는 의미는 둘 다 널 문자('\0')라는 뜻
	if (*b == '\0' || *a > *b) return 1;
	if (*a == '\0' || *a < *b) return -1;
	return 0;
}

// 리스트 전체를 출력하는 함수
void print_list(char* arr[], int size) {
	for (int i = 0; i < size; i++)
		printf("%s | ", arr[i]);
	printf("\n");
}

// 버블 소팅 방법으로 리스트를 오름차순으로 정렬하는 함수
void bubbleSort_ASC(char* arr[], int size) {

	for (int i = 1; i < size; i++) {
		for (int j = 0; j < size - i; j++) {

			// arr[j+1]가 arr[j]보다 사전 순으로 앞서 있을 때
			if (compareString(arr[j], arr[j + 1]) > 0) {
				// arr[j]와 arr[j+1] 스왑
				char* tmp;
				tmp = arr[j];
				arr[j] = arr[j + 1];
				arr[j + 1] = tmp;
			}
		}
	}
}
```

```c
// 버블 소팅 방법으로 리스트를 내림차순으로 정렬하는 함수
void bubbleSort_DESC(char* arr[], int size) {

	for (int i = 1; i < size; i++) {
		for (int j = 0; j < size - i; j++) {

			// arr[j]가 arr[j+1]보다 사전 순으로 앞서 있을 때
			if (compareString(arr[j], arr[j + 1]) < 0) {
				// arr[j]와 arr[j+1] 스왑
				char* tmp;
				tmp = arr[j];
				arr[j] = arr[j + 1];
				arr[j + 1] = tmp;
			}
		}
	}
}

char* dogBreeds[] = {
	"Yorkshire Terrier", "French Bulldog", "Golden Retriever",
"Dachshund",
	"Samoyed", "Cairn Terrier", "Shiba Inu", "Pomeranian",
	"Pomsky", "Australian Shepherd", "Bichon Frise", "Dalmatian",
"American Eskimo",
	"Chiweenie", "Brussels Griffon", "Papillon", "Chihuahua", "Toy
Poodle"
};

int main(void)
{

	printf("BUBBLE SORT\n");
	printf("\n-----------------------------------------\n\n");
	printf("[Input String] : ");
	print_list(dogBreeds, SIZE);

	printf("[Ascending order] : ");
	bubbleSort_ASC(dogBreeds, SIZE);
	print_list(dogBreeds, SIZE);

	printf("[Descending order] : ");
	bubbleSort_DESC(dogBreeds, SIZE);
	print_list(dogBreeds, SIZE);
	printf("\n-----------------------------------------\n\n");

	printf("Press \"Enter\" to quit the program...\n");
	getchar();
	return 0;
}
```

3. Insertion Sorting

Insertion Sorting 은 이름에서 알 수 있듯이 특정 위치의 아이템을 앞 리스트 사이에 삽입하며 정렬하는 방식이다. 여기서 앞 리스트란 전체 리스트 중 특정 아이템을 지정하고, 그 앞쪽에 위치한 아이템들의 집합을 말한다. 특정 아이템은 두 번째 아이템부터 차례대로 지정되며, 지정된 아이템을 앞 리스트의 정렬 상태를 유지할 수 있는 곳에 삽입한다. 따라서 이미 정렬된 리스트를 Insertion Sorting 을 사용하여 정렬한다면 지정한 아이템을 따로 삽입할 필요가 없기 때문에 시간 복잡도가 크게 감소한다. Insertion Sorting 또한 Bubble Sorting 과 마찬가지로 compareString 함수를 활용하여 문자열의 우선순위를 검사했다.

```
// 삽입 정렬 방법으로 리스트를 오름차순으로 정렬하는 함수
void insertionSort_ASC(char* arr[], int size) {
	for (int i = 1; i < size; i++) {
		int j = 0;
		char* tmp = arr[i];
		while (compareString(arr[j], tmp) < 0)
			j++; // 조건을 만족하는 j를 찾을 때까지 증가
		for (int k = i - 1; k > j - 1; k--)
			arr[k + 1] = arr[k]; // 자리를 한 칸 씩 당김
		arr[j] = tmp;
	}
}
```

변수 j는 지정된 아이템의 삽입할 위치를 결정하는 변수로 사용된다.

ii. 전체 C 언어 구현 코드

```c
#include <stdio.h>
#define SIZE 18

/* 사전순을 기준으로 문자열을 비교하여
* str1 > str2 일 때 1,
* str1 = str2 일 때 0,
* str1 < str2 일 때 -1을
* 반환하는 함수
*/
int compareString(char* a, char* b) {

	while (*a != '\0' && *b != '\0' && *a == *b) a++, b++;

	if (*a == *b) return 0;	// *a와 *b가 같다는 의미는 둘 다 널 문자('\0')라는 뜻
	if (*b == '\0' || *a > *b) return 1;
	if (*a == '\0' || *a < *b) return -1;
	return 0;
}

// 리스트 전체를 출력하는 함수
void print_list(char* arr[], int size) {
	for (int i = 0; i < size; i++)
		printf("%s | ", arr[i]);
	printf("\n");
}

// 삽입 정렬 방법으로 리스트를 오름차순으로 정렬하는 함수
void insertionSort_ASC(char* arr[], int size) {
	for (int i = 1; i < size; i++) {
		int j = 0;
		char* tmp = arr[i];
		while (compareString(arr[j], tmp) < 0)
			j++;
		for (int k = i - 1; k > j - 1; k--)
			arr[k + 1] = arr[k];
		arr[j] = tmp;
	}
}

// 삽입 정렬 방법으로 리스트를 내림차순으로 정렬하는 함수
void insertionSort_DESC(char* arr[], int size) {
	for (int i = 1; i < size; i++) {
		int j = 0;
		char* tmp = arr[i];
```

```
			while (compareString(arr[j], tmp) > 0)
				j++;
			for (int k = i - 1; k > j - 1; k--)
				arr[k + 1] = arr[k];
			arr[j] = tmp;
		}
}

char* dogBreeds[] = {
		"Yorkshire Terrier", "French Bulldog", "Golden Retriever",
"Dachshund",
		"Samoyed", "Cairn Terrier", "Shiba Inu", "Pomeranian",
		"Pomsky", "Australian Shepherd", "Bichon Frise", "Dalmatian",
"American Eskimo",
		"Chiweenie", "Brussels Griffon", "Papillon", "Chihuahua", "Toy
Poodle"
};

int main(void)
{
		printf("INSERTION SORT\n");
		printf("\n----------------------------------------\n\n");
		printf("[Input String] : ");
		print_list(dogBreeds, SIZE);

		printf("[Ascending order] : ");
		insertionSort_ASC(dogBreeds, SIZE);
		print_list(dogBreeds, SIZE);

		printf("[Descending order] : ");
		insertionSort_DESC(dogBreeds, SIZE);
		print_list(dogBreeds, SIZE);
		printf("\n----------------------------------------\n\n");

		printf("Press \"Enter\" to quit the program...\n");
		getchar();
		return 0;
}
```

III. 본론

이번 레포트를 통해 Bubble Sorting 과 Insertion Sorting 을 활용하여 문자열 배열을 오름차순, 내림차순으로 정렬해보았다. 문자열을 사전순으로 비교할 때 문자를 순서대로 비교하여 판단한다는 사실을 배웠다. 문자열을 정의할 때 char*을 사용했는데, char 배열을 사용해도 됐을 것 같다.