

# SP SEMINAR (C STUDY) ARRAY, POINTER & STRING

daybreaker@SPARCS

2007. 7. 18

# CONTENTS

- ◉ Array
- ◉ Pointer
- ◉ String

# ARRAY

- 하나의 이름으로 여러 개의 데이터 사용
- 선언 : `elementType arrayName[length];`
  - Index :  $0 \sim \text{length} - 1$
  - `arrayName[index]`로 사용
- Not resizable
  - 한번 만들어진 배열은 크기를 변경할 수 없음
  - 배열의 개별 원소를 바꾸는 것은 가능

# POINTER

- ◎ 변수 : 데이터 + 그 데이터의 위치
  - 데이터의 위치만을 별도로 다룰 수 있다.
  - 포인터도 변수이므로 포인터의 포인터도 가능
- ◎ 선언 : `dataType* pointerName;`
  - `void*`는 모든 종류의 변수를 가리킬 수 있음
- ◎ 변수의 주소 가져오기
  - `int a;`  
`int *p = &a;`
- ◎ 배열 이름은 포인터와 같다.
  - `Type of arrayName = elementType*`

# POINTER : 동적 할당

- ◎ Runtime에 사용할 메모리 공간을 결정
  - 배열 크기를 임의로 정하고 싶을 때
  - 임의 크기의 linked-list, stack, queue 구현

- ◎ `void* malloc(int size)`

- `int* myArray = (int*) malloc(sizeof(int) * 100);`  
`myArray[0] = 999; ...`

- ◎ `free(void*)`

- `free(myArray);`

# POINTER : 메모리 다루기

◎ **memcpy** (void\* src, void\* dest, int size);

- src에서 dest로 size만큼의 bytes를 복사
- Binary data 다룰 때 많이 사용
  - char buffer[]로부터 원하는 형식의 변수로 복사
  - 파일로부터 구조체 읽기
  - 네트워크 전송을 위한 packet 구성

※ Binary data란?

ASCII 문자 및 CP949/UTF-8 한글 등의 일반 'text'가 아닌, int, double 등의 메모리 데이터를 그대로 저장하는 것.

이렇게 저장된 binary file을 메모장과 같은 text editor로 열면 마구 깨진 글자들이 보이는 경우가 일반적이다. (exe, bmp, png, mp3 등등)

# PROBLEM #1

1. 임의의 개수의 줄 수를 가지고, 각 줄에는 숫자가 쓰여진 텍스트 파일을 읽는 프로그램을 작성해보자.
2. 다음 구조체를 그대로 파일에 쓰고 읽어보자.

```
struct student {  
    int id;  
    char name[30];  
    int birth_year;  
    unsigned char birth_month;  
    unsigned char birth_day;  
}
```

# THINK YOURSELF #1

- ◉ Pointer를 파일에 저장하고 다시 읽어올 수 있을까?
- ◉ 임의의 길이를 가지는 배열을 binary 형태로 저장하고 다시 읽어오려면 어떻게 해야 할까?

# STRING

- ◉ 배열의 특수한 형태
  - `elementType = char`

- ◉ 유의 사항

- 코드에 직접 작성한 문자열은 변경 불가
  - `char* myString = "ABCDEFGH";`
- `malloc` 등으로 할당하여 만든 문자열은 변경 가능
  - `char* myString = (char*) malloc(10);`  
`strcpy("ABCDEFGH", myString);`

# STRING : 함수

◎ **strcpy** (const char\* src, const char\* dst);

- src의 내용을 dst로 복사

◎ **strcat** (const char\* src, const char\* dst);

- src에 dst를 덧붙임
  - null 제거 후 이어서 대입하고 끝에 null 붙임

◎ **strlen** (const char\* str)

- 문자열 길이 리턴 (null 문자까지의 문자 수)

◎ **strn\***

- buffer overflow 방지

※ Buffer overflow란?

지정된 변수의 길이 제한을 벗어난 메모리 영역까지 침범하여 write하는 것. null 문자로 막혀있지 않은 문자열인 경우 주로 발생한다.

# STRING : 함수

- ⊙ **strcmp** (const char \*s1, const char \*s2)
  - 문자열 2개를 비교
  - s2보다 s1이 작으면 음수, 크면 양수, 같으면 0
  
- ⊙ char\* **strtok** (char \*s, const char \*delim)
  - 문자열을 delim에 따라 분리해냄
  - 가급적 사용하지 않는 것이 좋다.
    - s의 포인터를 수정해버림
    - multi-thread를 지원하지 않음

## PROBLEM #2

- 텍스트 파일로부터 각 줄을 원소로 가지는 문자열 배열을 만들어보자.
  - 각각의 원소는 임의의 길이의 문자열을 담을 수 있어야 하며, 프로그램 종료 시 적절하게 메모리를 해제하여야 한다.
  - 텍스트 파일의 줄 별 문자열의 최대 길이는 임의로 가정해도 되지만, 버퍼는 동적으로 할당하라.

# ADDITIONAL PROBLEMS

- 임의의 길이를 가지는 stack과 queue를 구현해보자. (type은 int로 가정)
  - push(int data), int pop()
  - enqueue(int data), int dequeue()
- 임의의 길이를 가지는 linked list를 구현해보자.
  - first(), last(), remove(int index), append(int data)

※ stack, queue, linked list

가장 기초적인 자료 구조. 구체적인 정의와 예제 등은 Wikipedia를 검색해보기 바란다.

# THINK YOURSELF #2

- 임의의 개수의 key, value 쌍을 가지며, key를 통해 그 key에 해당하는 value를 가져올 수 있는 자료 구조를 Dictionary라고 한다.

이를 구현할 수 있는 방법을 생각해보자.

- key 검색을 *효율적으로* 하려면 어떻게 하는 것이 좋을까?