

CS SCHOOL

문자와 문자열

문자열 3가지 표현 방법

```
#include <stdio.h>
```

```
#include <string.h>
```

```
int main(int argc, char * argv[])  
{
```

```
    char str1[6] = "abcde"; //문자열_1
```

```
    char * str2 = "abcde"; //문자열 2
```

```
    char * str3 = (char*)malloc(sizeof(char) * 6); //문자열_3  
    strcpy(str3, "abcde");
```

문자열 표현의 3가지 방법

이 세가지 방법의 특징과 의미를
알면 문자열은 끝!!

문자와 문자열

문자열 첫 번째 유형

```
char str1[6] = "abcde";//문자열_1
```

```
int lenOfStr = strlen(str1);
```

```
for (int i = 0; i < lenOfStr; i++)  
    str1[i] = 'b';
```

```
printf("%s\n", str1);
```

char형 문자의 배열을 stack에 할당

위와 같은 문법은 선언 과 초기화를 함께 한 경우에만 쓸 수 있습니다.

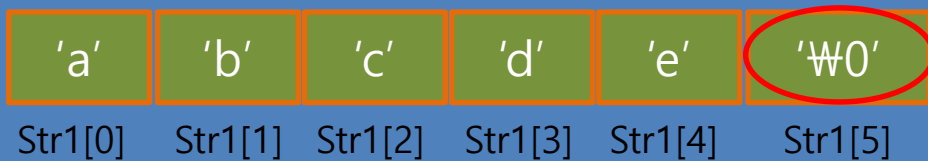
문자와 문자열

문자열 첫 번째 유형

```
char str1[6] = "abcde"; // 문자열_1
```

Stack 영역

null 문자 : 문자열의 끝을 나타냄.



```
char str1[6];  
str1 = "abcde";
```

선언 후 초기화는 안됩니다!! (X)

```
for (int i = 0; i < lenOfStr; i++)
```

```
    str1[i] = 'b';
```

배열요소에 접근해 변경 가능!!

문자와 문자열

문자열 첫 번째 유형

```
char str[] = "I am your \0 father!";
```

```
printf("%s \n", str);
```

도중에 '\0'이 들어갔습니다.
어디까지 출력될까요??



문자열을 출력하면

'\0'(null 문자)를 만날 때까지 출력합니다.

문자와 문자열

표준 라이브러리 <string.h>

1. strlen()

- 1) 기능 : 문자열의 길이를 반환
'\0'은 길이에서 제외!

2. strcpy()

- 1) 기능 : 문자열을 복사
문자열은 대입연산자로 복사할 수 없습니다!!

3. strcat()

- 1) 기능 : 문자열을 이어 붙임.

4. strcmp()

- 1) 기능 : 문자열이 서로 같은지 비교

문자와 문자열

Strlen() 함수의 용법

1) 인터페이스

```
size_t strlen(const char * s);
```

문자열 길이 정보 반환

반환 길이가 '\0'을 제외한 길이

2) 사용 방법

```
char str[] = "abcde"; 맨 마지막의 \0 절대 잊지 말 것!!
```

```
int length = strlen(str); null 문자 제외이므로 5
```

```
printf("length : %d \n", length);
```

문자와 문자열

Strcpy(), strncpy() 함수의 용법

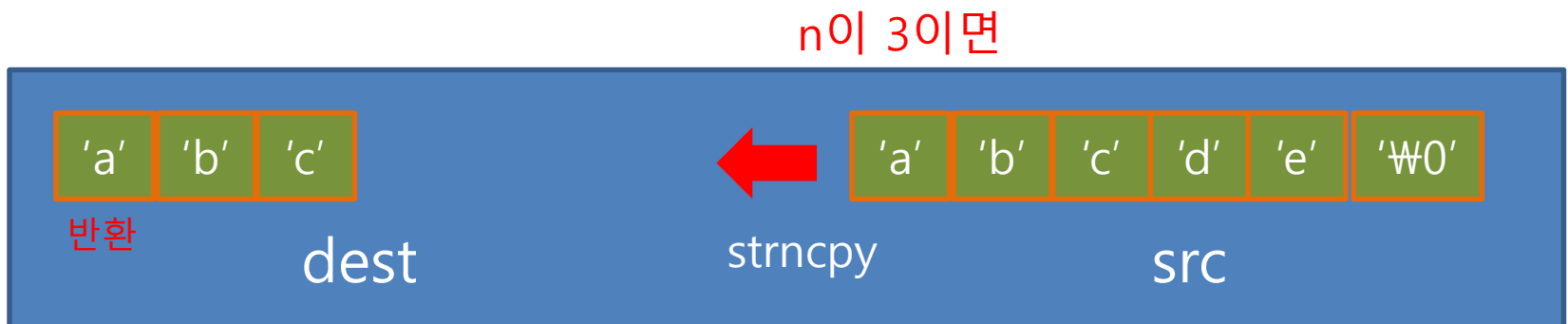
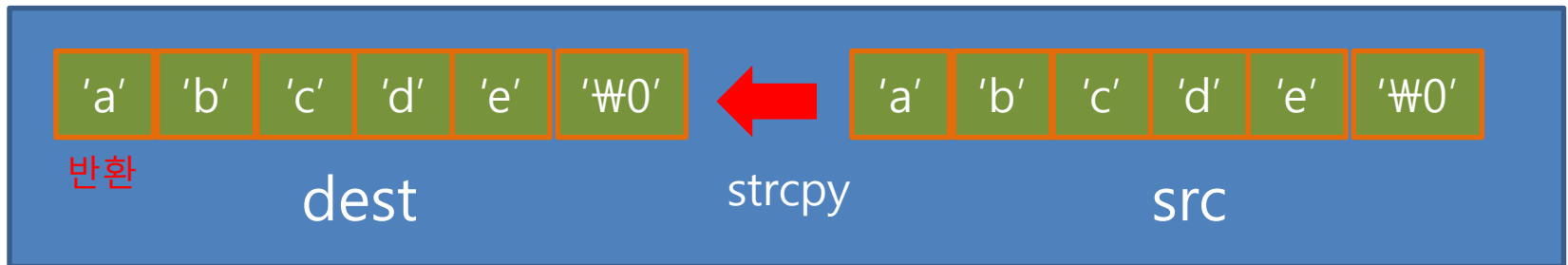
1) 인터페이스

```
char * strcpy(char * dest, const char * src);
```

```
char * strncpy(char * dest, const char * src, size_t n);
```

src의 문자열을 dest에 복사
반환형은 dest의 주소값 반환

복사할 문자열의 크기가 n보다 크면
n 만큼만 복사



'\0'이 빠진 잘못된 문자열

문자와 문자열

Strcpy(), strncpy() 함수의 용법

2) 사용 방법 `strcpy()`

```
char str1[30];  
char str2[30] = "abcde";
```

```
strcpy(str1, str2);
```

```
printf("%s \n", str1);
```

`strncpy()`

```
char str1[30];  
char str2[30] = "abcde";
```

```
strncpy(str1, str2, 3);
```

```
printf("%s \n", str1);
```



C언어가 인정하지 않는
'\0' 문자가 빠진 문자열

문자와 문자열

Strcpy(), strncpy() 함수의 용법

2) 사용 방법

```
const int MAX = 3;  
char str1[30];  
char str2[30] = "abcde";
```

```
strncpy(str1, str2, MAX);
```

```
str1[MAX] = '\0';
```

 null 문자 삽입하여 문자열 완성

```
printf("%s \n", str1);
```



문자와 문자열

문자열 두 번째 유형

```
char * str2 = "abcde";//문자열_2
```

```
int lenOfStr = strlen(str2);
```

```
for (int i = 0; i < lenOfStr; i++)
```

```
    str2[i] = 'b';
```

이 부분에서 에러가 납니다!
왜 에러가 날까요??

```
printf("%s\n", str2);
```

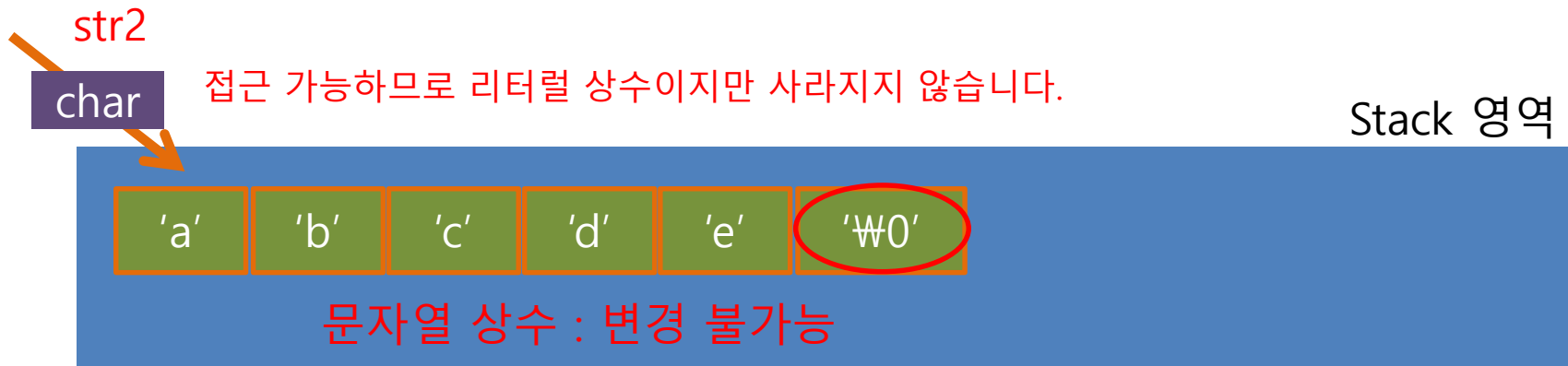
여기서 "abcde"는 문자열 상수입니다! 즉 변경 불가능!!!

문자열 상수 "abcde" 는 str2라는 포인터로 접근 가능하므로
다음 행에서 사라지지 않습니다!

문자와 문자열

문자열 두 번째 유형

```
char * str2 = "abcde";//문자열_2
```



```
for (int i = 0; i < lenOfStr; i++)  
    str2[i] = 'b';  
    str2 + i
```

접근은 가능하지만
변경은 불가능(X)

문자와 문자열

문자열 세 번째 유형

```
char * str3 = (char*)malloc(sizeof(char) * 6); //문자열_3  
strcpy(str3, "abcde");
```

```
int lenOfStr = strlen(str3);
```

```
for (int i = 0; i < lenOfStr; i++)  
    str3[i] = 'b';
```

```
printf("%s\n", str3);
```

```
free(str3);  
str3 = 0;
```

Heap 영역에 메모리를 먼저 할당한 후
strcpy()함수를 이용해 문자열을 복사합니다.

문자와 문자열

malloc()과 free()

```
const int ARRAY_LENGTH = 5;
//heap 영역에 할당~~
int * arrPtr = (int *)malloc(sizeof(int) * ARRAY_LENGTH);
```

Heap 영역에 할당

```
for (int i = 0; i < ARRAY_LENGTH; i++)
    arrPtr[i] = i + 1;
```

포인터는 배열처럼 접근 가능

```
for (int i = 0; i < ARRAY_LENGTH; i++)
    printf("arrPtr[%d] = %d \n", i, arrPtr[i]);
```

//다 쓰고 난 후 꼭 해제해주세요!

```
free(arrPtr);
```

Heap 영역에서 해제

```
arrPtr = 0;
```

null 포인터로 해줍니다.(최대한 안전하게 코딩)

문자와 문자열

malloc()과 free()

//heap 영역에 할당~~

```
int * arrPtr = (int *)malloc(sizeof(int) * ARRAY_LENGTH);
```

void*를 반환하므로
원하는 형으로 형 변환 해야 함.

byte 단위로 할당!!

$4 * 5 = 20$ byte

//다 쓰고 난 후 꼭 해제해주세요!

```
free(arrPtr);
```

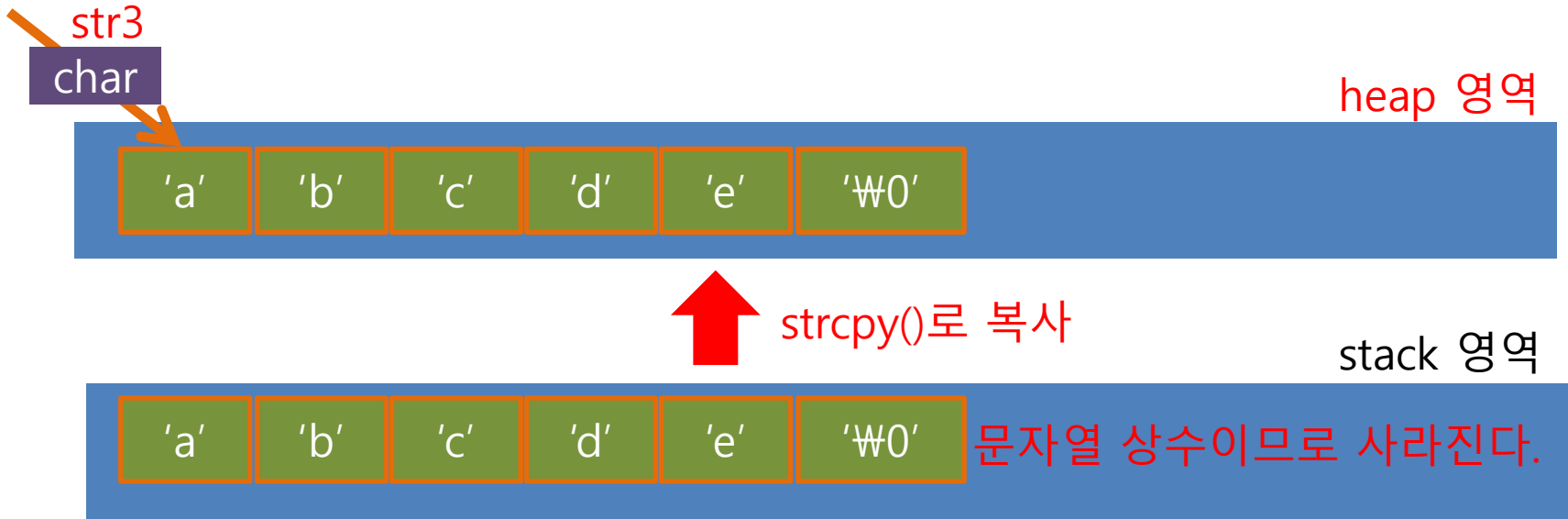
 해제 안 해주면 계속 남아 있음.

```
arrPtr = 0;
```

문자와 문자열

문자열 세 번째 유형

```
char * str3 = (char*)malloc(sizeof(char) * 6); //문자열_3  
strcpy(str3, "abcde");
```



```
for (int i = 0; i < lenOfStr; i++)  
    str3[i] = 'b';
```

str3+i 배열처럼 접근 가능
 변경 가능

Call-by-Reference VS. Call-by-Value

Swap() 함수 구현

```
void Swap1(int num1, int num2)
{
    int temp = num1;
    num1 = num2;
    num2 = temp;
}
```

두 함수 모두 두 개의 매개변수 값을 서로 바꾸고 있습니다.
그런데..... 두 함수의 차이 아시겠습니까?

```
void Swap2(int * ptr1, int * ptr2)
{
    int temp = *ptr1;
    *ptr1 = *ptr2;
    *ptr2 = temp;
}
```

Call-by-Reference VS. Call-by-Value

Swap() 함수 구현

```
int num1 = 555;  
int num2 = 1;
```

```
Swap1(num1, num2);
```

```
printf("num1 : %d \n", num1);  
printf("num2 : %d \n", num2);
```

Call by Value
: 매개변수로 값을 전달하면
stack 영역에서 매개변수를 복사
변수에 접근할 수 없으므로
값 변경을 불가능하다.

```
int num1 = 555;  
int num2 = 1;
```

```
Swap2(&num1, &num2);
```

Call by Reference
: 매개변수로 포인터를 전달하면
변수에 바로 접근해 값을 변경할 수 있다.

```
printf("num1 : %d \n", num1);  
printf("num2 : %d \n", num2);
```

재귀함수

재귀함수란?

```
long long SumToNumber(int num)
{
    //탈출조건
    if (num == 1) 탈출조건
        return 1;

    return SumToNumber(num - 1) + num;
}
```

같은 이름의 함수가 함수 내에 나올 때 이를 재귀함수라 부릅니다.

재귀함수

실습 예제 : 1부터 1억까지의 수를 모두 더하는 프로그램

세 가지 알고리즘으로 설계를 해봅시다!

```
//1. recursion 재귀함수
```

```
sum = SumToNumber(num);
```

```
//2. for 문 for 문
```

```
for (int i = 0; i <= num; i++)  
    sum += i; 가장 일반적으로 떠올릴 알고리즘
```

```
//3. 등차수열 등차수열
```

```
sum = ArithmeticSequence(num);
```

재귀함수

1. 재귀함수를 이용한 방법

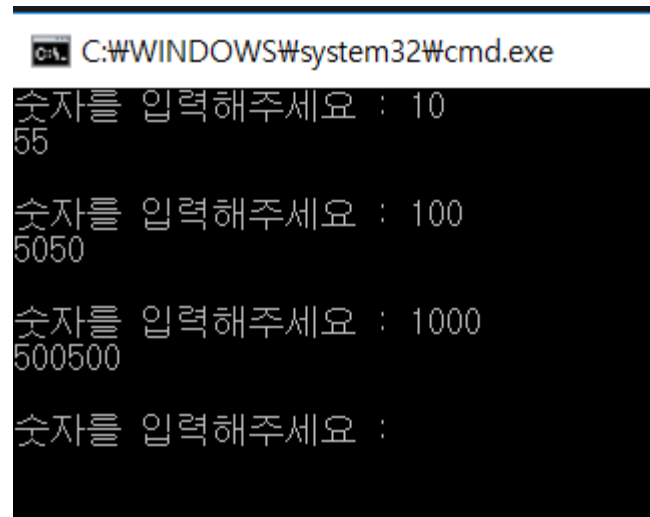
```
long long SumToNumber(int num)
{
    //탈출조건
    if (num == 1)
        return 1;

    return SumToNumber(num - 1) + num;
}
```

재귀함수

1. 재귀함수를 이용한 방법

10, 100, 1000까지는 잘 돌아갑니다



```
C:\WINDOWS\system32\cmd.exe
숫자를 입력해주세요 : 10
55
숫자를 입력해주세요 : 100
5050
숫자를 입력해주세요 : 1000
500500
숫자를 입력해주세요 :
```

여기에 100,000,000 을 입력해 봅시다.

재귀함수

Stack overflow

Microsoft Visual Studio



Unhandled exception at 0x00CD1739 in Recursion.exe: 0xC00000FD: Stack overflow (parameters: 0x00000001, 0x00CE2F74).

Stack overflow, 즉 스택이 터졌습니다!!

☐ Break when this exception type is thrown

[Break and open Exception Settings](#)

Break

Continue

Ignore

재귀함수

for 문을 이용한 방법 vs 등차 수열을 이용한 방법

두 가지 알고리즘 모두 잘 돌아갑니다

```
//2. for 문  
for (int i = 0; i <= num; i++)  
    sum += i;  
  
//3. 등차수열  
sum = ArithmeticSequence(num);
```

성능의 측면에서는 어떨까요?

재귀함수

for 문을 이용한 방법

10,000,000 을 입력하면 1억 번 돌아야 겠네요.....

```
for (int i = 0; i <= num; i++)  
    sum += i;
```

재귀함수

등차 수열을 이용한 방법

$$1 + 2 + + num$$

등차 수열 공식

n : 항의 개수
a : 첫째 항
l : 마지막 항

$$\frac{n(a + l)}{2} = \frac{(num) * (1 + num)}{2}$$

```
long long ArithmeticSequence(long long num)
{
    long long sum = (num * (1 + num)) / 2;
    return sum;
}
```

수식 한번 계산 하면 끝이네요!!!!