



제1장

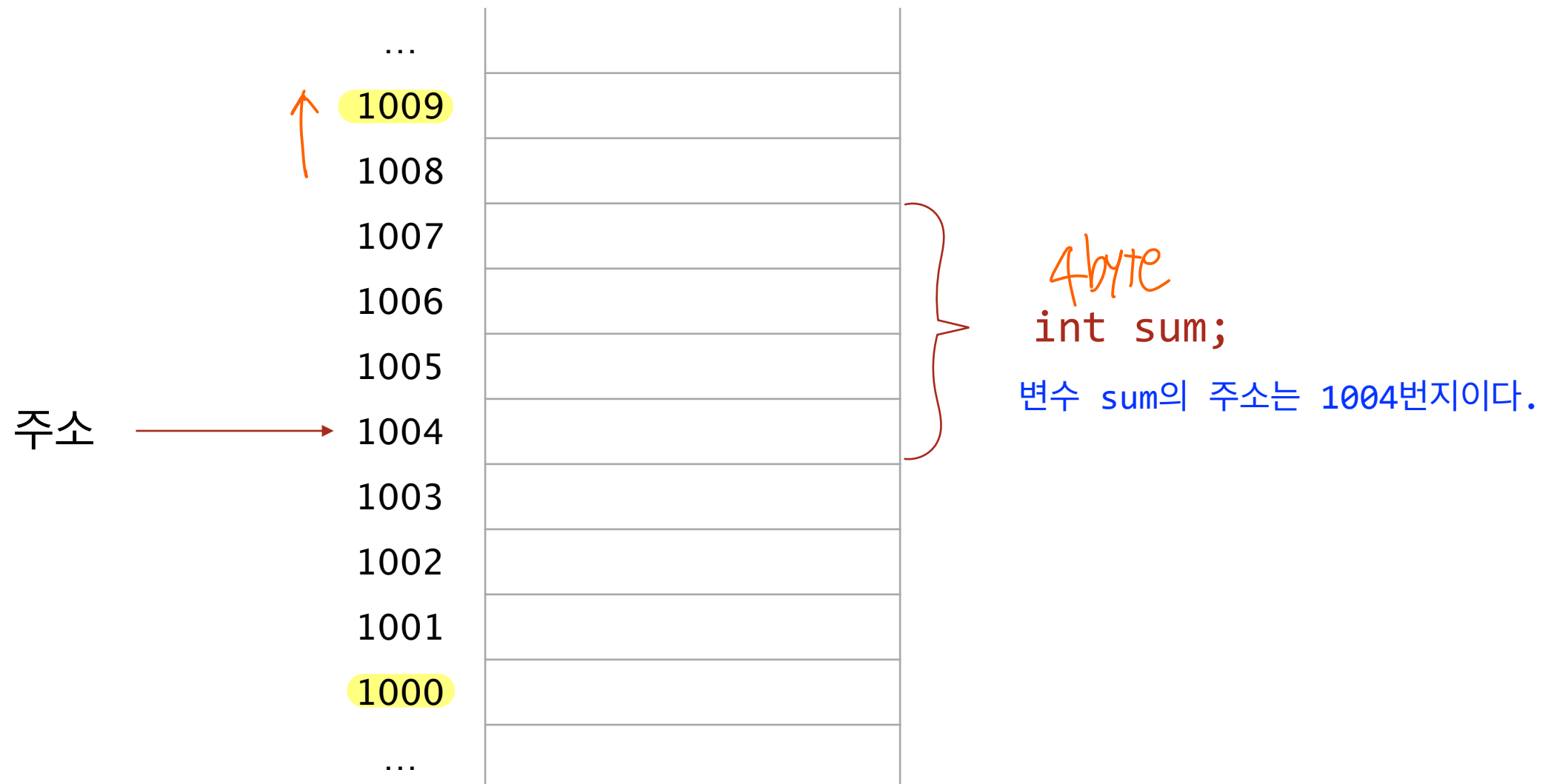
C 언어 리뷰

기초 문법 리뷰

배열, 포인터, 문자열, 동적메모리할당

메모리

- 컴퓨터의 메모리는 데이터를 보관하는 장소
- 바이트(8 bits) 단위로 주소가 지정됨
- 모든 변수는 주소를 가짐



포인터

정수를 값으로 → int
문자를 값으로 → char

- 포인터(pointer)는 메모리 주소를 값으로 가지는 변수이다. 포인터 변수는 다음과 같이 선언된다.

`type-name * variable-name;`

- `variable-name`은 선언된 포인터 변수의 이름이며, `*`는 `variable-name`이 포인터 변수임을 표시하고, `type-name`은 포인터 변수 `variable-name`에 저장될 주소에 저장될 데이터의 타입을 지정

* 만으로 변수의 타입을 알 수 있는 항.

`int * ptr;`

정수형 포인터변수

포인터

- 연산자 ^{주소 연산자} &는 변수로부터 그 변수의 주소를 추출하는 연산자이다.

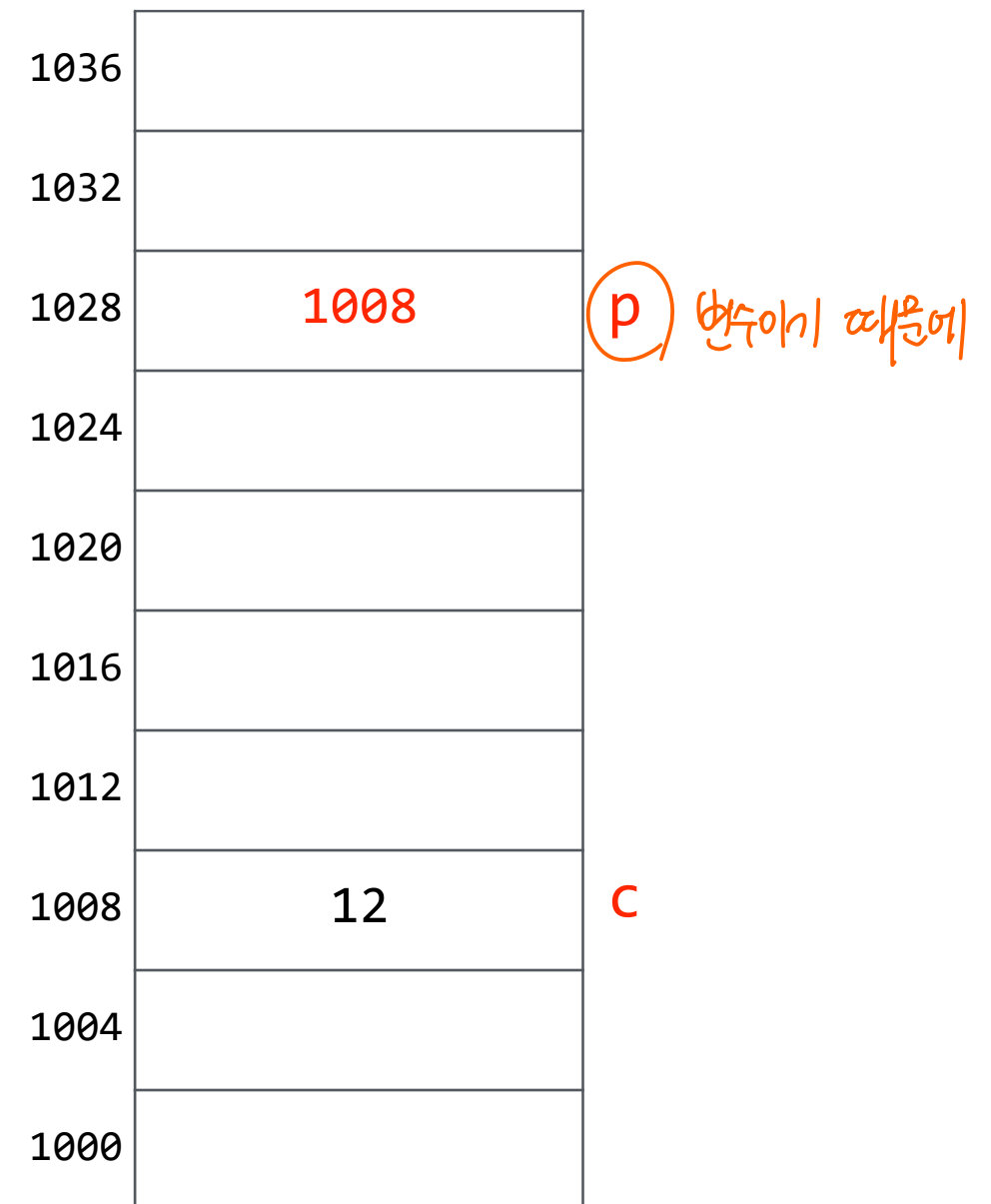
```
int c = 12;
```

```
int *p; 정수형 포인터 변수
```

```
p = &c;
```

^{= 1008}

포인터변수 p에 변수 c의
주소를 저장한다.



포인터

`int x=1, y=2;`
`int *ip;`
`ip = &x;`
`y = *ip;`
`*ip = 0;`

선언에서 포인터라는 것을 알림
 간접 참조 연산자
 오른쪽에 있는 ip가 가리키는 변수의 값
 왼쪽에 있는 ip가 가리키는 변수의 저장공간

| | | |
|------|------|----|
| 1036 | | |
| 1032 | | |
| 1028 | 1 | x |
| 1024 | 2 | y |
| 1020 | | |
| 1016 | | |
| 1012 | | |
| 1008 | 1028 | ip |
| 1004 | | |
| 1000 | | |

```
int x=1, y=2;
int *ip;
ip = &x;
```

| | | |
|------|------|----|
| 1036 | | |
| 1032 | | |
| 1028 | 1 | x |
| 1024 | 1 | y |
| 1020 | | |
| 1016 | | |
| 1012 | | |
| 1008 | 1028 | ip |
| 1004 | | |
| 1000 | | |

`y = *ip;`

| | | |
|------|------|----|
| 1036 | | |
| 1032 | | |
| 1028 | 0 | x |
| 1024 | 1 | y |
| 1020 | | |
| 1016 | | |
| 1012 | | |
| 1008 | 1028 | ip |
| 1004 | | |
| 1000 | | |

`*ip = 0;`

포인터와 배열

- 포인터와 배열은 매우 긴밀히 연관되어 있다.
- 예를 들어 다음과 같이 선언된 배열 a가 있다고 하자.

```
int a[10];
```

배열의 첫 번째 칸의 주소

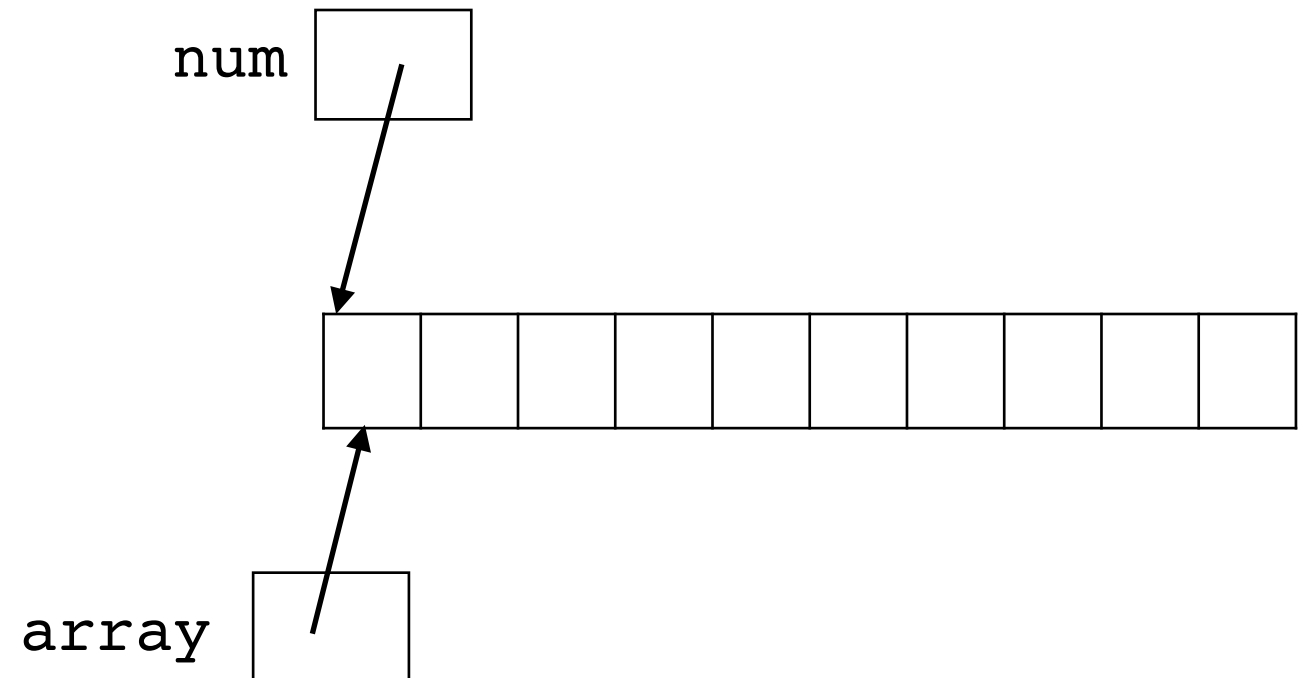
배열의 이름은
배열의 시작 주소를 저장하
는 포인터 변수임
(단 그 값을 변경할 수 없음)



예제

```
#include <stdio.h>
int main(void)
{
    int sum, average;
    int num[10];
    for ( int i = 0; i < 10; i++ )
        scanf("%d", &num[i]);
    sum = calculate_sum( num );
    average = sum / 10;
    printf("%d\n", average);
    return 0;
}

int calculate_sum( int *array )
{
    int sum = 0;
    for ( int i = 0; i < 10; i++ )
        sum = sum + array[i];
    return sum;
}
```



배열을 매개변수로 받을 때
`int array[]`
대신 이렇게 포인터로 받을 수도 있다.

포인터 arithmetic

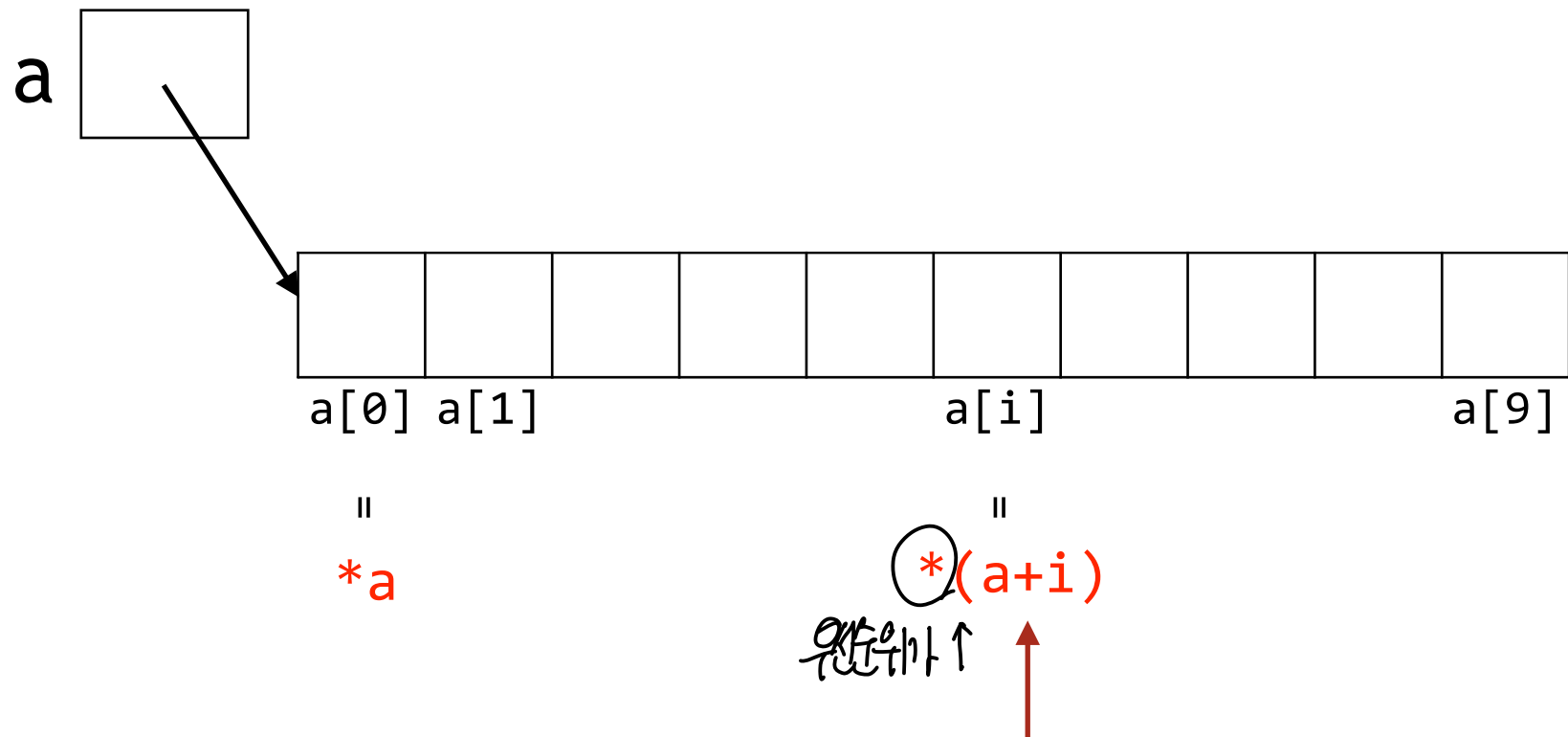
- $*a$ 와 $a[0]$ 은 동일한 의미이다.
- 또한 $a[1]$ 은 $*(a+1)$ 과 동일하고, $a[i]$ 는 $*(a+i)$ 와 동일하다.

인덱싱이 동일한 의미

+

(X) $a+1 = 1001$
(O) $a+1 = 1004$

```
int a[10];
```

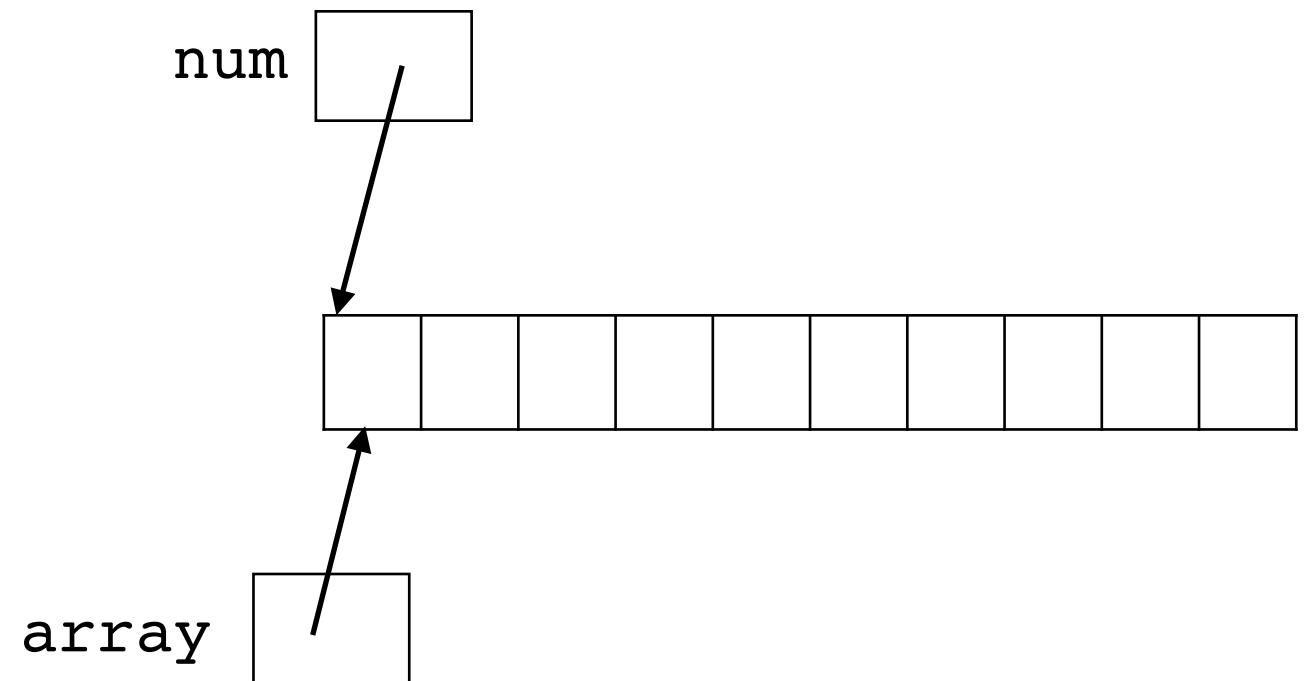


이런 연산을 포인터 arithmetic이라고 부른다.

포인터 arithmetic

```
#include <stdio.h>
int main(void)
{
    int sum, average;
    int num[10];
    for ( int i = 0; i < 10; i++ )
        scanf("%d", &num[i]);
    sum = calculate_sum( num );
    average = sum / 10;
    printf("%d\n", average);
    return 0;
}
```

```
int calculate_sum(int * array)
{
    int sum = 0;
    for ( int i = 0; i < 10; i++ )
        sum = sum + *(array + i);
    return sum;
}
```



`sum = sum + *(array + i);` ← `sum = sum + array[i]`와 동일하다.

동적메모리 할당

- 변수를 선언하는 대신 프로그램의 요청으로 메모리를 할당할 수 있다. 이것을 **동적 메모리 할당**(dynamic memory allocation)이라고 부른다.
- malloc 함수를 호출하여 동적메모리할당을 요청하면 요구하는 크기의 메모리를 할당하고 그 **시작 주소를 반환**한다.

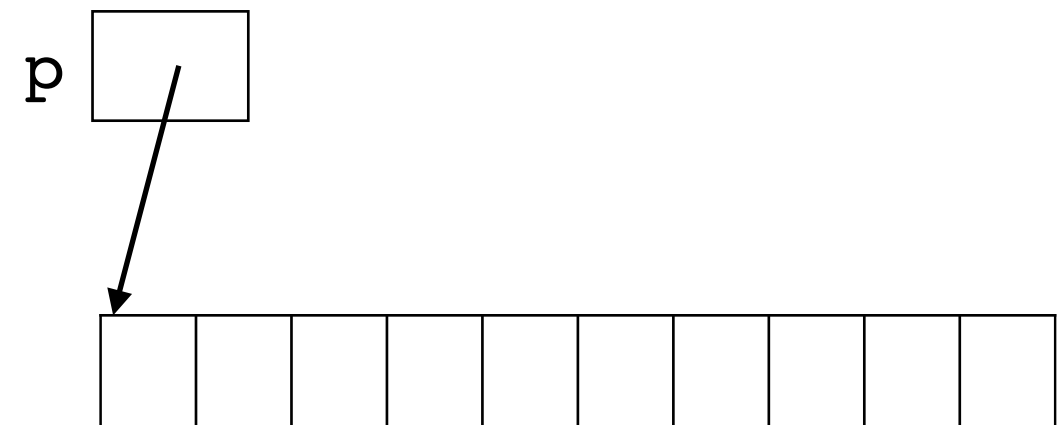
malloc 함수

malloc이 반환하는 주소는 타입이 없는 주소(void *)이다. 정수들을 저장하기 위해서 이것을 int *로 변환한다. 반드시 필요한 건 아니다.

행렬한 데이터가 이걸 알기 위해
포인터 변수에 저장

```
int * p;  
p = (int *) malloc(40);  
if (p==NULL) { 매우 예외적인 상황  
/* 동적 메모리 할당이 실패 */  
/* 적절한 조치를 취한다. */  
}
```

할당받을 메모리의 크기를 byte단위로 지정한다. 여기서는 10개의 정수를 저장하기 위해서 40바이트를 요청하였다.



길이가 10인 배열처럼

```
p[0] = 12;  
p[1] = 24;  
*(p+2) = 36;  
...
```

malloc으로 할당받은 메모리는 이렇게 보통의 배열처럼 사용한다.

▶ 이렇게 malloc으로 할당 받은 메모리는 나중에 불필요해지면 반드시 free 함수를 이용하여 반환해야 한다. 이 점에 대해서는 다음 장에서 상세히 다룬다.

배열 키우기

- 동적으로 할당된 배열은 공간이 부족할 경우 더 큰 배열을 할당하여 사용할 수 있다.

수능

int가 4 byte일 수도 있고 다른 수도 있음.

크기 한정

배열이름은 수정 X
int array[4];

int * array = (int *) malloc(4 * sizeof(int)); / 1000

/* 배열 array의 크기가 부족한 상황이 발생한다. */

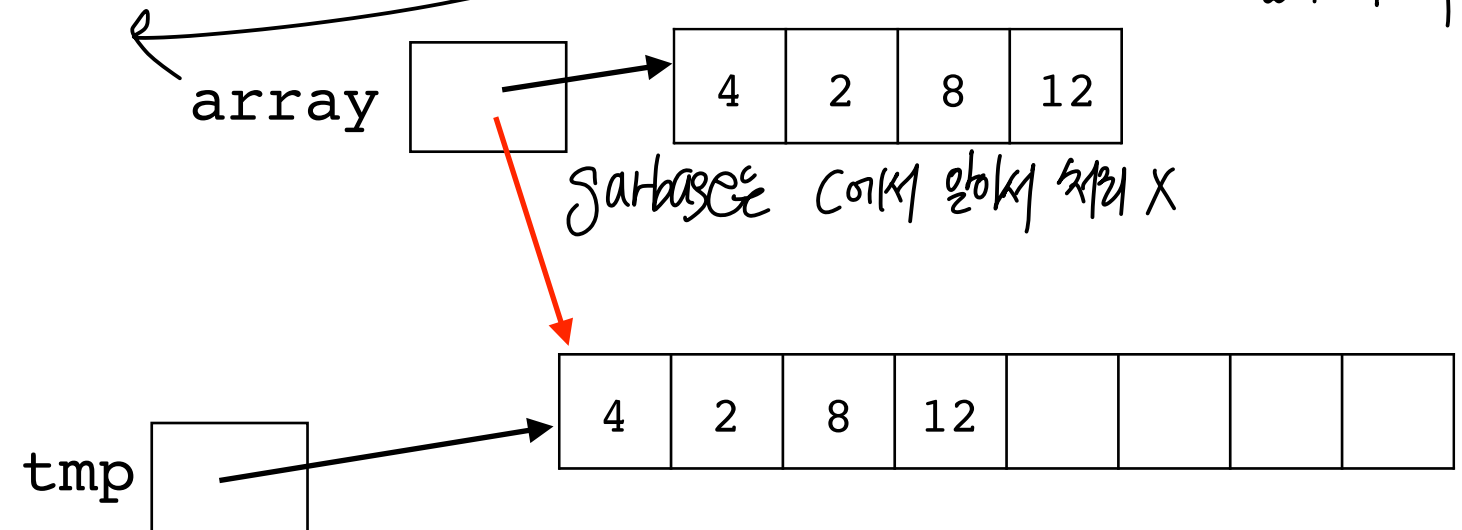
int * tmp = (int *) malloc(8 * sizeof(int)); 2000

for (int i=0; i<4; i++)
tmp[i] = array[i];

array = tmp;

free(array);

다음 공간은 다르게 사용하기 위해
있기 때문에



문자열 (string)

- 문자열은 char타입의 배열의 각 칸마다 문자 하나씩 저장됨

```
char str[6];  
str[0] = 'h';  
str[1] = 'e';  
str[2] = 'l';  
str[3] = 'l';  
str[4] = 'o';  
str[5] = '\0';
```

널 문자
null character(' \0')는 문자열의 끝을 표시하는 역할을 한다.
즉 배열의 크기가 문자열의 길이보다 적어도 1만큼 길어야 한다.

str

| | | | | | |
|---|---|---|---|---|----|
| h | e | l | l | o | \0 |
|---|---|---|---|---|----|

- C 언어는 문자열을 생성하는 편리한 방법을 제공

C 언어 문법은 아님.

char str[] = "hello";
↓ 컴파일러
문자배열
혹은

char *str = "hello";

하지만 이렇게 정의된 문자열은 수정이 불가능하다는 점에서 위의 두 방법과 다르다. 이것을 string literal이라고 부른다.

문자열 첫 번째 문자의 주소

수정 불가

문자열은 컴파일 시점에서 별도로 보관되고 첫 번째 문자의 주소로 바뀜

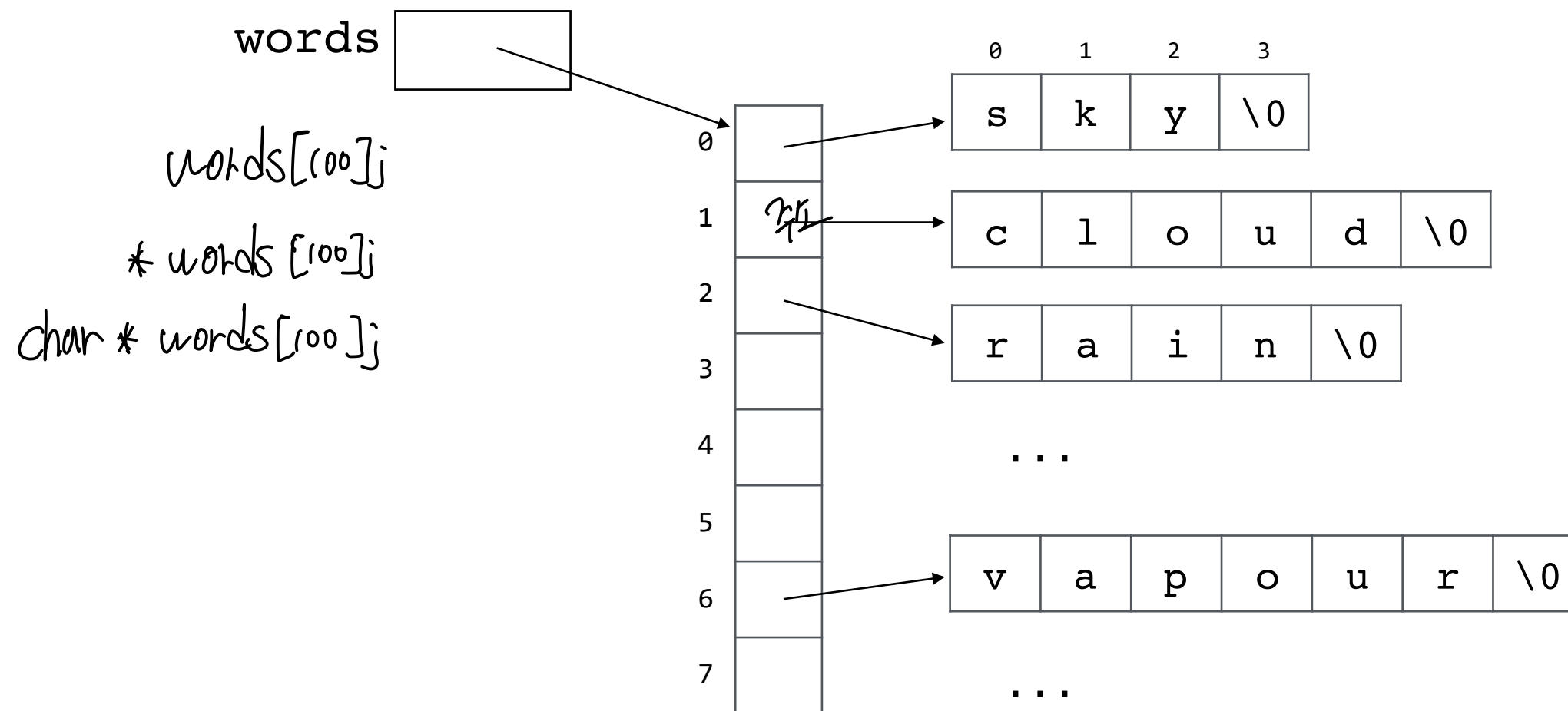
string.h 라이브러리 함수

- string.h 라이브러리는 문자열을 다루는 다양한 함수를 제공

| | |
|--------|---------|
| strcpy | 문자열 복사 |
| strlen | 문자열의 길이 |
| strcat | 문자열 합치기 |
| strcmp | 문자열 비교 |

문자열들의 저장

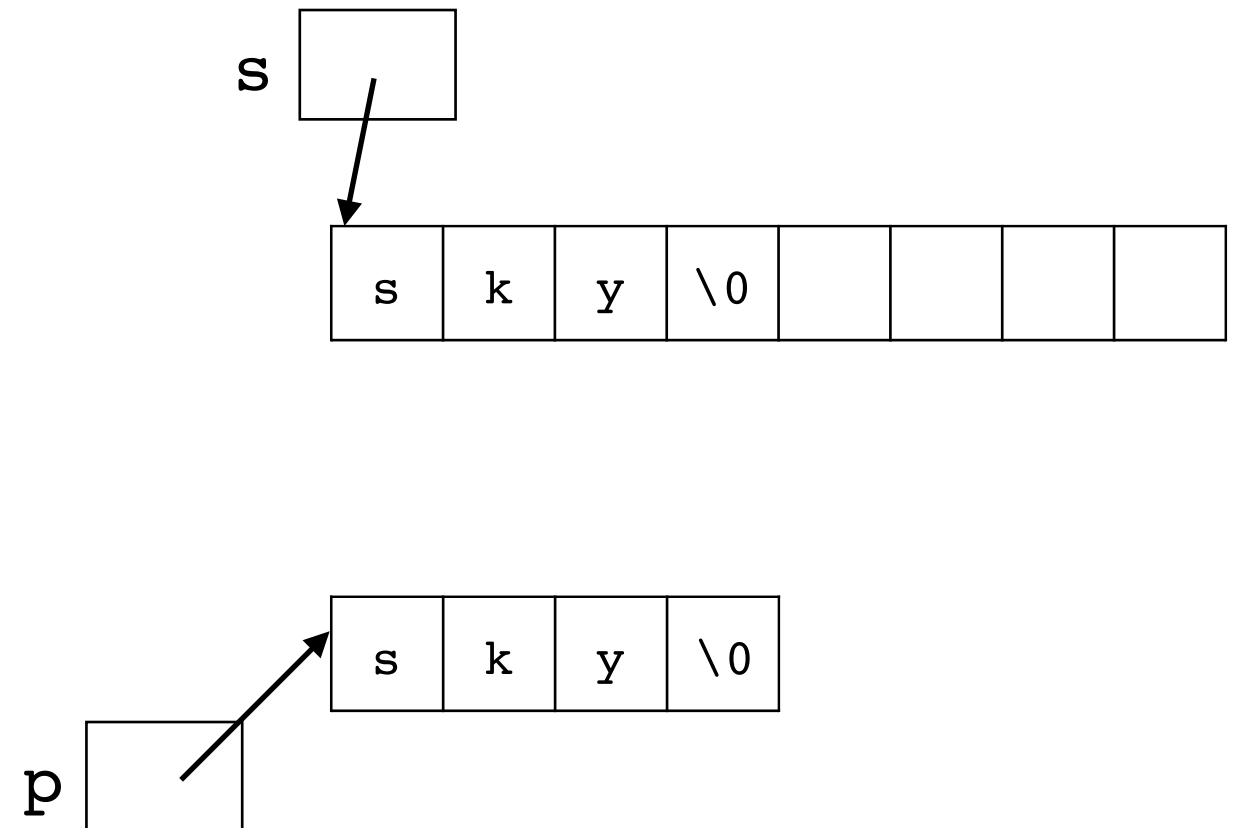
- 여러 개의 단어들을 포인터를 이용하여 아래 그림과 같이 저장해보자.



문자열 복사: strdup

매개변수로 받은 하나의 문자열을 복제하여 반환한다.

```
char * strdup(char *s)
{
    char *p;
    p = (char *)malloc(strlen(s)+1);
    if (p != NULL)
        strcpy(p, s);
    return p;
}
```



strcpy와의 차이는 ?

파일로부터 읽기

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

```
int main() {  
    FILE *fp = fopen("input.txt", "r");  
    char buffer[100];  
    while (fscanf(fp, "%s", buffer) != EOF)  
        printf("%s ", buffer);  
    ✓ fclose(fp);  
}
```

파일 포인터 (pointing to FILE *)
읽기 모드 ("r")

파일 읽고 쓰기

```
#include <stdio.h>

int main() {
    FILE * in_fp = fopen("input.txt", "r");
    FILE * out_fp = fopen("output.txt", "w");
    char buffer[100];
    while (fscanf(in_fp, "%s", buffer) != EOF)
        fprintf(out_fp, "%s ", buffer);
    fclose(in_fp);
    fclose(out_fp);
}
```

실습 문제

연습 1

프로그램을 실행하면 화면에 프롬프트(\$)와 한 칸의 공백문자를 출력하고 사용자의 입력을 기다린다.

↓
\$ hello↵ ← 문장을 입력하고 리턴(↵) 키를 친다.

hello:5 ← 리턴(↵) 을 제외하고 입력한 문장을 그대로 출력하고 입력한 문장의 길이를 출력한다.

\$ welcome to the class↵ 공백문자

welcome to the class:20 ← 공백문자도 포함하여 카운트한다.

\$ programming is fun, right? ↵

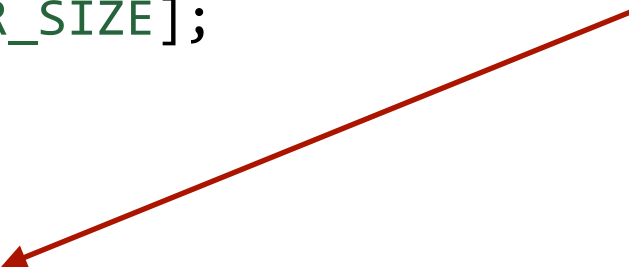
programming is fun, right? :35 ← 문장의 앞뒤에 붙은 공백까지 그대로 출력해야 한다.

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

#define BUFFER_SIZE 100

int main() {
    char buffer[BUFFER_SIZE];
    while (1) {
        printf("$ ");
        scanf("%s", buffer);
        printf("%s:%d\n", buffer, strlen(buffer));
    }
    return 0;
}
```

프롬프트를 출력한다.



gets

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

```
#define BUFFER_SIZE 10
```

← 배열의 크기보다 더 긴 문장을 입력해본다.
어떤 문제가 생기는지 확인한다.

```
int main() {
```

```
    char buffer[BUFFER_SIZE];
```

```
    while (1) {
```

```
        printf("$ ");
```

```
        gets(buffer);
```

라인단위

← gets 함수는 라인을 통채로 읽는다.

```
        printf("%s:%d\n", buffer, strlen(buffer));
```

```
    }
```

```
    return 0;
```

```
}
```

Warning : unsafe

버퍼 사이즈를 초과해서 gets가 읽어버릴 수 있음.

fgets

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

```
#define BUFFER_SIZE 100
```

```
int main() {
```

```
    char buffer[BUFFER_SIZE];
```

```
    while (1) {
```

```
        printf("$ ");
```

```
        fgets(buffer, BUFFER_SIZE, stdin);
```

```
        printf("%s:%d\n", buffer, strlen(buffer));
```

```
    }
```

```
    return 0;
```

```
}
```

stdin은 표준 입력, 즉 키보드를 의미한다.

메모리 액세스 위반

엔터키까지 버퍼에 저장함.
엔터키까지 길이에 포함됨.
틀바꿈

문제가 생기는 이유와 해결방법은 ?

$buffer[strlen(buffer) - 1] = '\0'$

20 넘었을 때 인원이 못가 안됨.

getchar

```
#include <stdio.h>
#include <string.h>
#define BUFFER_SIZE 100

int main() {
    char buffer[BUFFER_SIZE];
    int k;
    while (1) {
        printf("$ ");
        k = read_line(buffer, BUFFER_SIZE);
        printf("%s:%d\n", buffer, k);
    }
    return 0;
}

int read_line( char str[], int limit )    {
    int ch, i = 0;
    while ((ch = getchar()) != '\n')
        if (i < limit-1)
            str[i++] = ch;
    str[i] = '\0';
    return i;
}
```

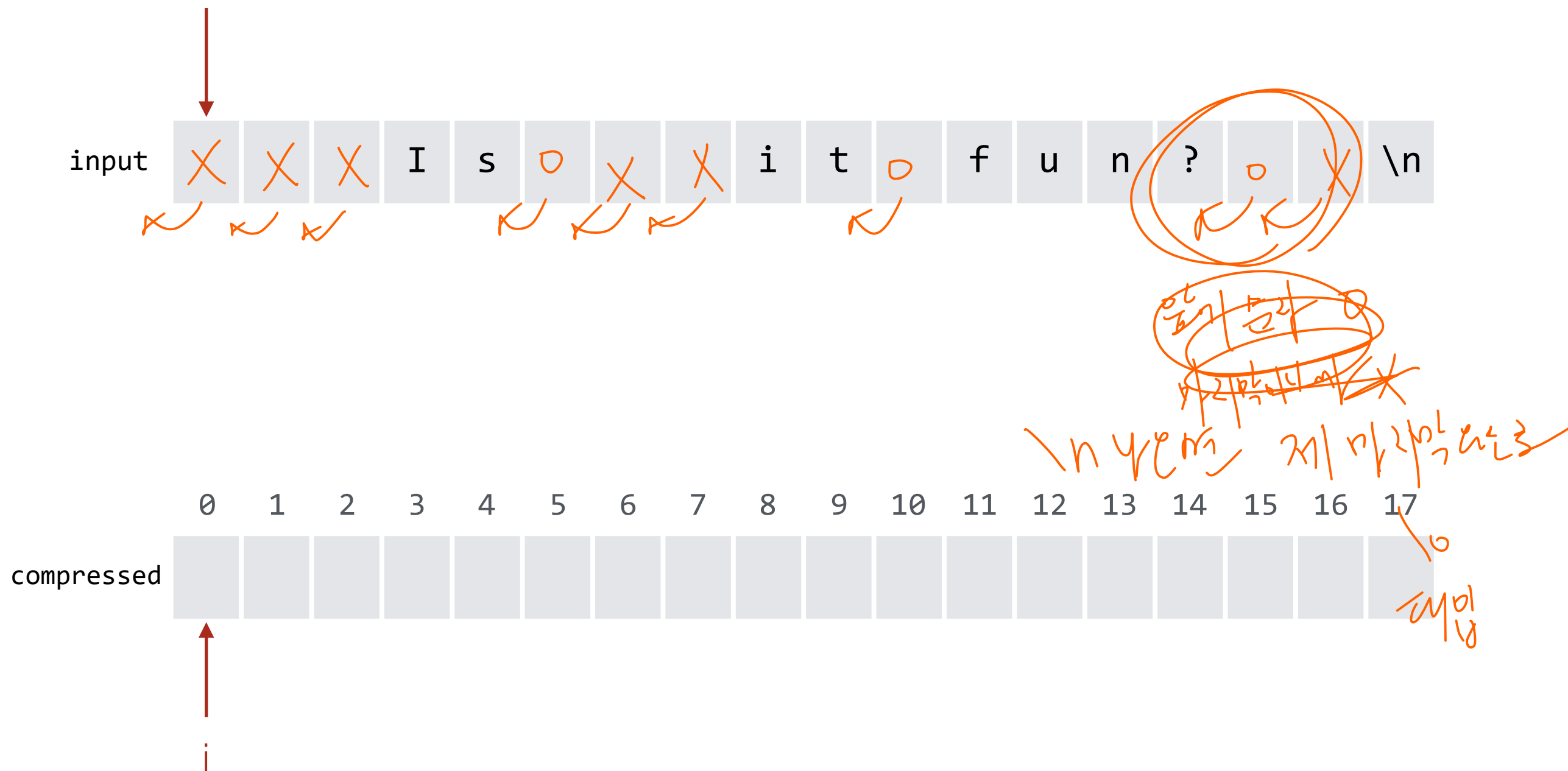
연습 2

공백문자들이 문장의 앞, 중간, 뒤에 포함되어 있다.

```
$    hello ↵  
hello:5  
$ welcome  to the class ↵  
welcome to the class:20 ↵  
$  programming is    fun, right?  ↵  
programming is fun, right?:26 ↵
```

문장의 앞과 뒤에 붙은 공백문자들은 제거하고
단어 사이에 두 개 이상의 연속된 공백문자들은 하나의 공백 문자로 대체하라.

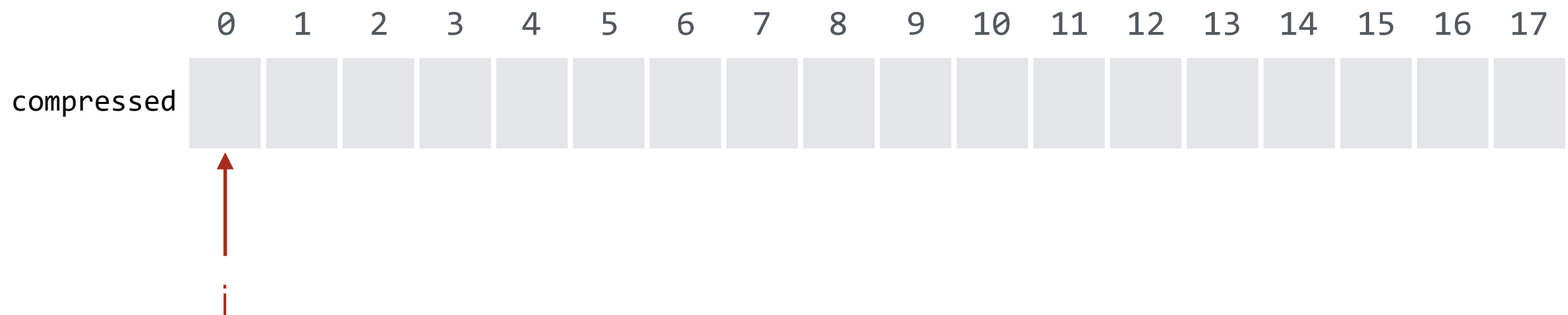
압축하기



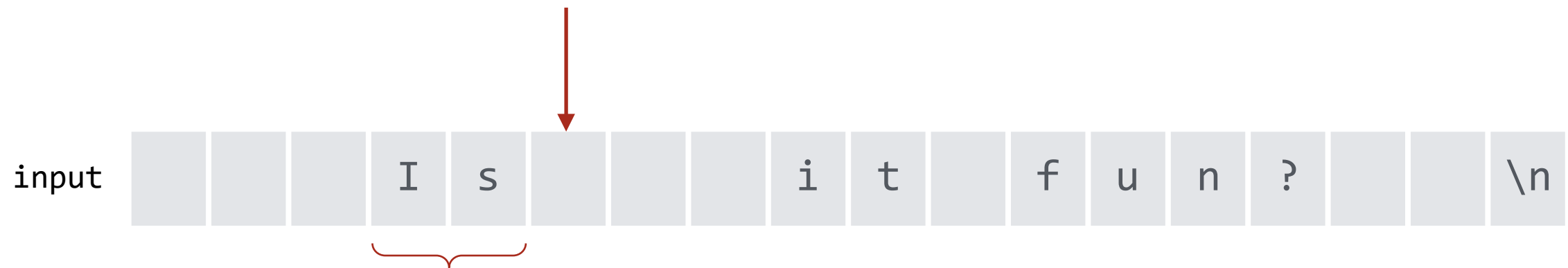
압축하기



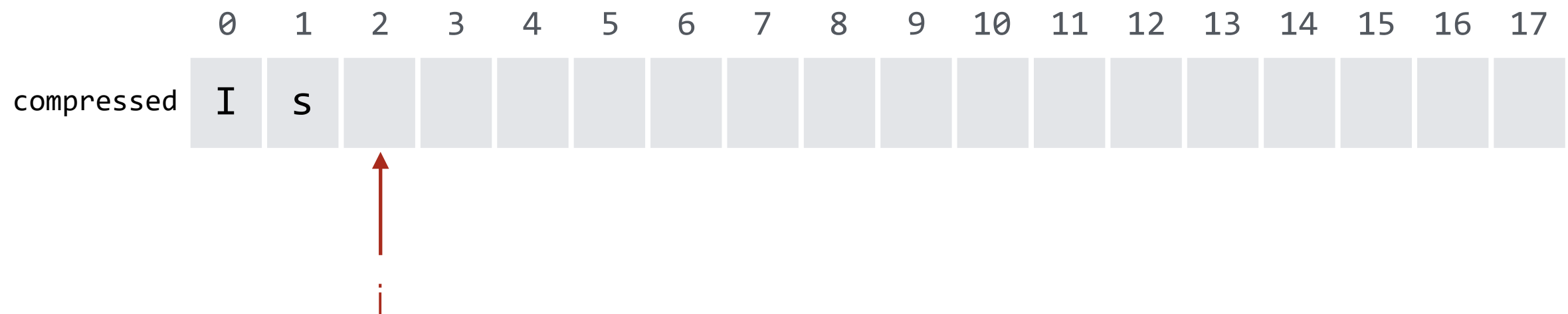
맨 앞의 공백들을 건너뛴다.



압축하기



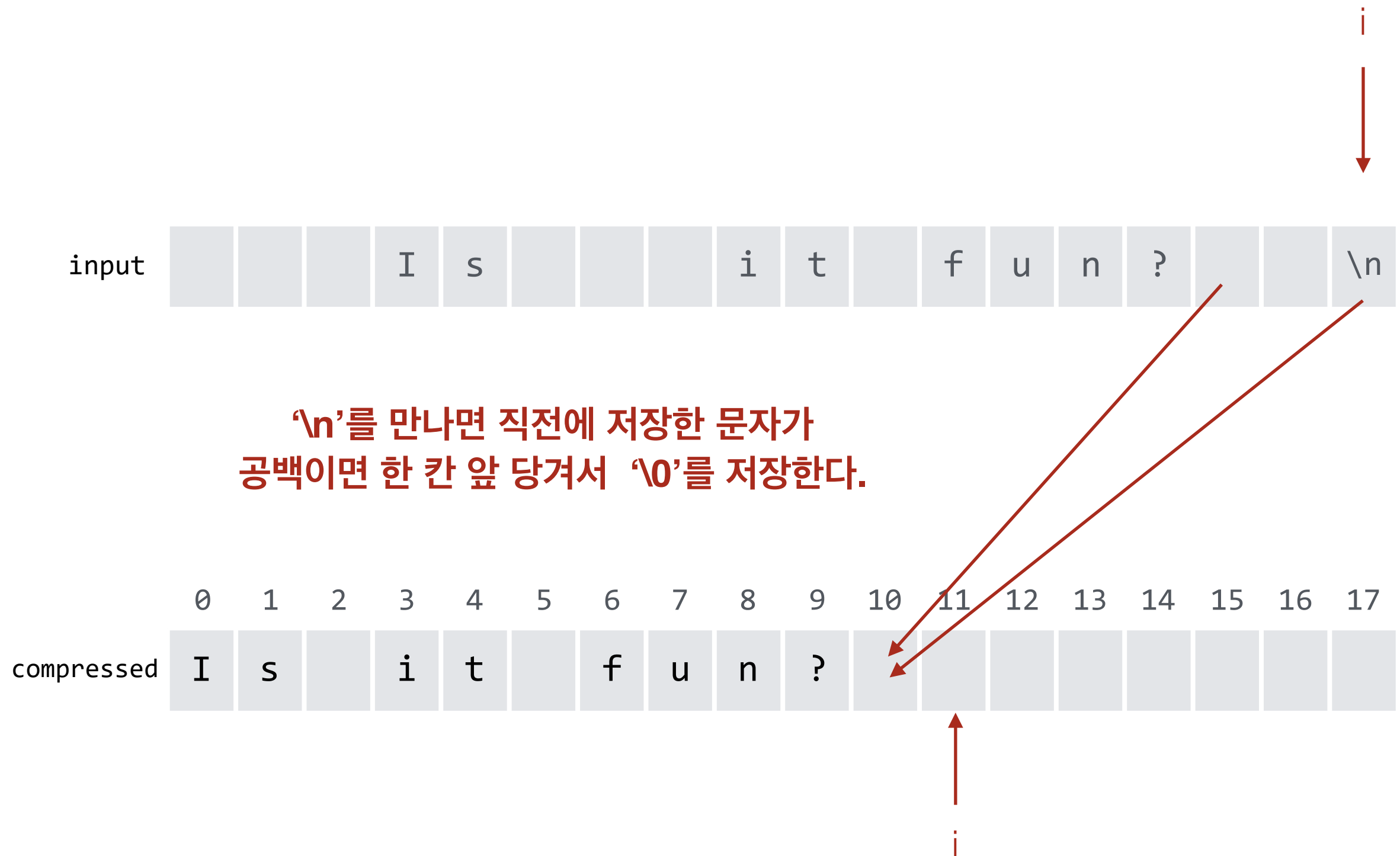
공백이 아니면 저장한다.



압축하기



압축하기



compress while reading

```
#include <ctype.h> ← white space 문자인지 검사하는 isspace 함수를 제공

int main() {
    char line[80];
    while(1) {
        printf("$ ");
        int length = read_line_with_compression(line, 80);
        printf("%s:%d\n", line, length);
    }
    return 0;
}

int read_line_with_compression( char compressed[], int limit ) {
    int ch, i = 0;
    while ((ch = getchar()) != '\n') {
        if (i < limit-1 && (!isspace(ch) || i == 0 무시 i > 0 && !isspace(compressed[i-1])))
            compressed[i++] = ch; 공백만 아니라 읽음
    }
    if (i > 0 && isspace(compressed[i-1])) ← 마지막 문자가 공백이면
        i--;
    compressed[i] = '\0';
    return i;
}
```

C++

C++에서의 동적 메모리 할당

- C++에서는 new 연산자와 new [] 연산자를 이용하여 동적 메모리 할당을 한다. 배열을 생성할 때는 new [] 연산자를 사용한다.

```
int * p;
```

```
p = new int [10];
```

 ← 10개의 정수를 저장하기 위한 배열을 생성하고, 배열의 시작 주소를 반환한다.

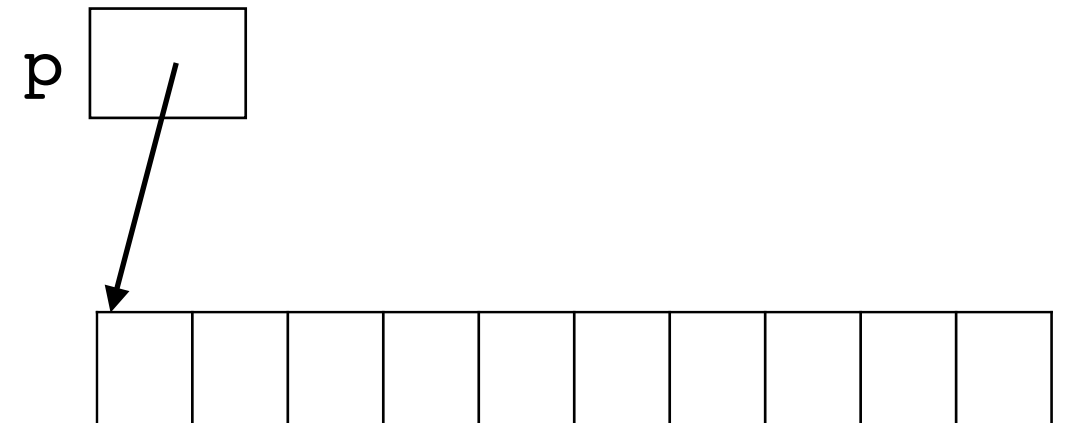
```
p[0] = 12;
```

```
p[1] = 24;
```

```
*(p+2) = 36;
```

```
...
```

← 이렇게 보통의 배열처럼 사용한다.



▶ new 혹은 new []로 할당 받은 메모리는 나중에 불필요해지면 반드시 delete 혹은 delete[]로 반환해야 한다. 이 점에 대해서는 다음 장에서 상세히 다룬다.

배열 키우기(array reallocation)

- 동적으로 할당된 배열은 공간이 부족할 경우 더 큰 배열을 할당하여 사용할 수 있다.

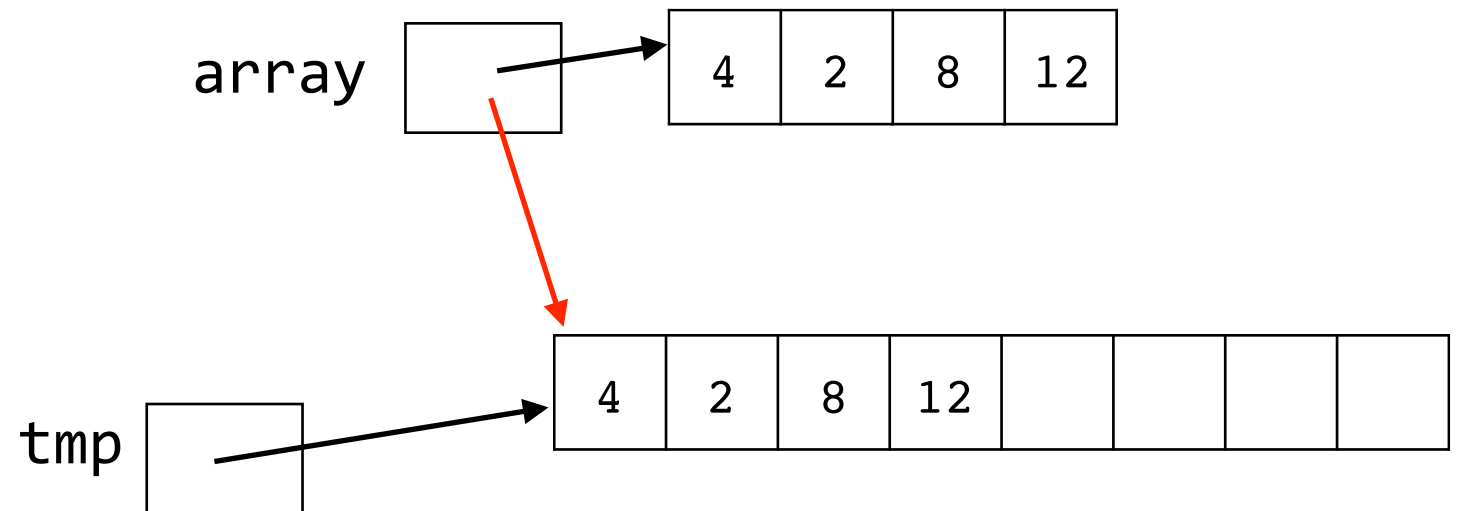
```
int capacity = 4;  
int *array = new int [capacity];
```

```
/* 배열 array의 크기가 부족한 상황이 발생한다. */
```

```
int *tmp = new int [capacity*2];
```

```
for (int i=0; i<capacity; i++)  
    tmp[i] = array[i];
```

```
array = tmp;  
capacity *= 2;
```



- C++에서는 두 가지 방식으로 문자열을 다룬다.
 - C 스타일의 문자 배열
 - `string` 클래스로 표현된 문자열

C++ string 클래스

```
#include <iostream>
#include <string>
using namespace std;
```

```
int main () {
```

```
    string s1("hello");
    string s2 = "hello";
    string s3 = string("hello");
    string s4{"hello"};
```

```
    cout << s1 << endl;
    cout << s2 << endl;
    cout << s3 << endl;
    cout << s4 << endl;
```

```
    return 0;
```

```
}
```

← 다양한 방법으로 string을 생성할 수 있다.

C++ string 클래스

```
#include <iostream>
using namespace std;

int main () {
    string str1 = "Hello";
    string str2 = "World";
    string str3;
    int len ;

    // copy str1 into str3
    str3 = str1;
    cout << "str3 : " << str3 << endl;

    // concatenates str1 and str2
    str3 = str1 + str2;
    cout << "str1 + str2 : " << str3 << endl;

    // total length of str3 after concatenation
    len = str3.size();
    cout << "str3.size() : " << len << endl;

    // compare two strings using ==
    string str4 = "Hello";
    if (str1 == str4)                // or, if (str1.compare(str4) == 0)
        cout << "Same" << endl;

    return 0;
}
```

C 스타일 문자열과 C++string간의 상호변환

```
#include <iostream>
using namespace std;

int main () {
    string str ("Welcome to the string world in C++ !!!");

    char cstr[100];                // 혹은 char *cstr = new char [100];

    strcpy(cstr, str.c_str()); ← strcpy 함수를 사용하기 위해서 C-style 문자열로 먼저 변환해주었다.

    cout << cstr << endl;

    string str2(cstr); ← C-style 문자열을 복사하여 C++ string을 만든다.

    cout << str2 << endl;

    return 0;
}
```

- ▶ c_str()함수는 새로운 문자열 데이터를 생성하지는 않으며, string 객체가 가진 문자열의 주소를 반환한다.
- ▶ 따라서 `const char *cstr = str.c_str()`을 하면 str과 cstr은 실제 문자열 데이터를 공유하게 된다.

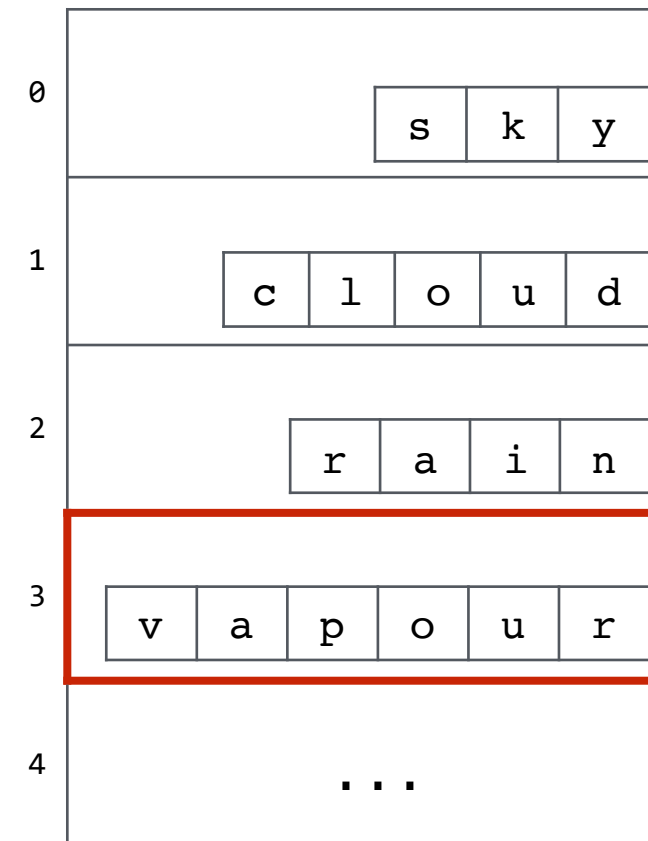
단어들 입력받아 저장하기

string 배열이다.

```
string words[MAXWORDS];  
int nwords = 0;
```

```
while (nwords < MAXWORDS) {  
    cin >> words[nwords];  
    if (words[nwords].compare("exit") == 0)  
        break;  
    nwords++;  
}
```

words



이 각각이 하나의 string 객체이다. 단순히 문자열 데이터 이외에도 다른 추가적인 데이터를 가진다.

← words[nwords] == "exit"으로 해도 된다.

- ▶ C++에서는 보통 이렇게 하는 것이 일반적이다. 물론 배열 대신 `std::vector<std::string>`를 사용할 수도 있다.

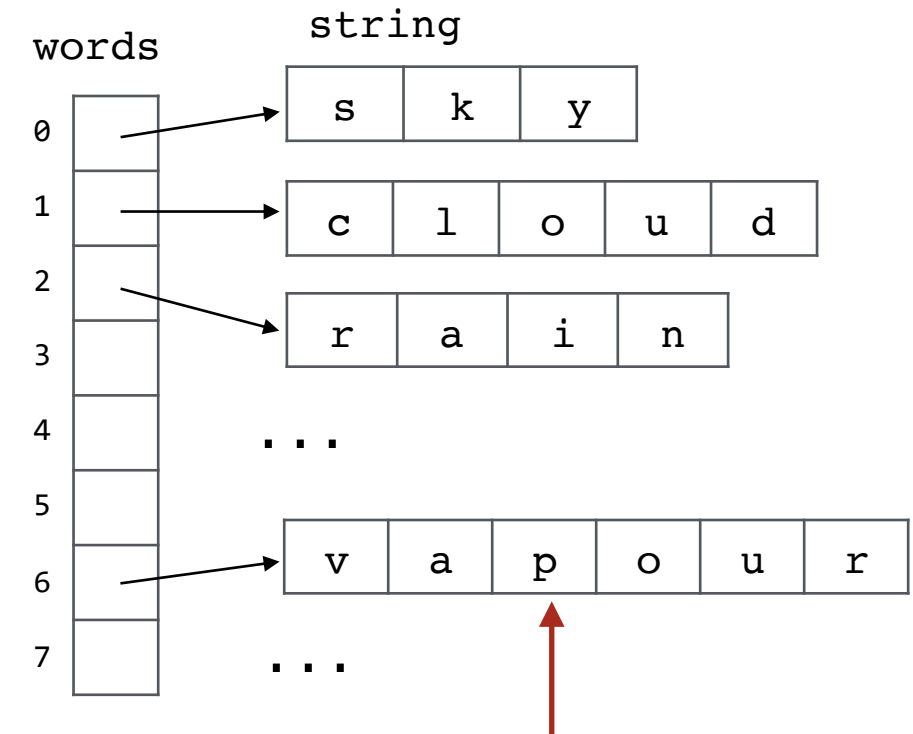
단어들 입력받아 저장하기

words는 string 포인터 배열이다.

```
string *words[MAXWORDS];  
int nwords;
```

```
string tmp;  
nwords = 0;
```

```
while (nwords < MAXWORDS) {  
    cin >> tmp;  
    if (tmp == "exit")  
        break;  
    words[nwords++] = new string(tmp);  
}
```



이 각각은 C style의 문자배열이 아닌 string 객체이다.

← copy constructor에 의해 string tmp를 복사한 새로운 string이 생성되고, 그 주소가 배열에 저장된다.

- ▶ 이렇게 객체들의 배열이 아닌 객체들의 포인터의 배열을 만드는 것은 C++ 보다는 C 스타일이라고 할 수 있다.

```
#include <iostream>
#include <fstream>
using namespace std;

int main() {
    ifstream in_file("input.txt");
    ofstream out_file("output.txt");
    string str;
    while (!in_file.eof()) { ← input.txt 파일의 끝에 도달했는지 검사한다.
        in_file >> str;
        out_file << str;
    }
    in_file.close();
    out_file.close();

    return 0;
}
```

- ▶ input.txt 파일을 아무렇게나 만든 후 생성된 output.txt 파일을 확인해보자.
- ▶ 두 파일을 동일하게 만들려면 어떻게 해야 할까?

```
#include <iostream>
using namespace std;
```

```
int main() {
    string line;
    while (1) {
        cout << "$ ";
```

```
        getline(cin, line);
```

← '\n' 문자를 만날 때 까지 라인 전체를 읽어준다.

```
        if (line == "exit")
            break;
```

```
        cout << line << ":" << line.size() << endl;
```

```
    }
```

```
    return 0;
```

```
}
```

```
int main() {
    while(1) {
        cout << "$ ";
        string str = read_line_with_compression();
        if (str == "exit") break;
        cout << str << ":" << str.size() << endl;
    }
}

string read_line_with_compression() {
    char ch;
    string str;

    while (1) {
        cin.get(ch);          // cin >> ch not works because it skips whitespaces.
        if (ch == '\n') break;
        if (!isspace(ch) || str.size() > 0 && !isspace(str.back()))
            str += ch;
    }
    if (str.size() > 0 && isspace(str.back()))
        str.back() = '\0';
    return str;
}
```