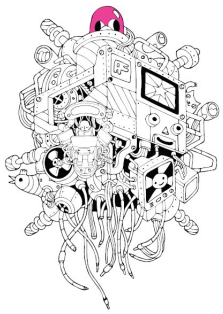


Chapter 21. 문자와 문자열 관련 함수

Chapter 21-0. 문자열 복습

character의 집합체

- character string (문자열)의 정의
  - 일련의 문자
- 표기 방법
  - 문자열 앞 뒤에 인용부호 " "를 이용
    - 문자열 상수
  - 문자와 문자열과의 차이

"C language"

"C" ← 문자가 하나라도 문자열 상수임

'C' ← char 유형 (1 바이트)인 문자 상수임

한 글자 저장 시
char c = 'a';

c라는 문자(char) 방 1 byte에는 a라는 문자가 저장됨

3

character string의 저장

- character array (문자 배열)의 이용 : 변수 형태의 문자열
  - 문자열 상수 "C language"를 저장하는 문자열 배열 c2[ ]를 선언하는 문장
    - 정확한 배열의 크기는 문자열의 길이보다 1이 크기 때문에 배열의 크기를 기술하지 않는 방법

```
char c2[ ] = "C language";
```

c2[]

C		l	a	n	g	u	a	g	e	\0
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]

  - 문자열의 마지막 NULL 문자를 포함한 정확한 크기를 기술하는 방법

```
char c2[11] = "C language";
```

4

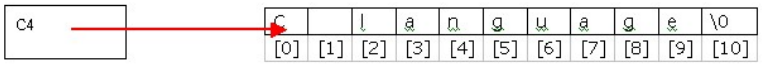
character string의 저장

- character array (문자 배열)의 이용 (계속)
  - 문자의 배열 초기 값 표현으로 대입 방법
    - 주의할 사항은 문자 배열 마지막에 반드시 직접 NULL 문자를 넣어야 함

```
char c1[ ] = {'C', ' ', 'l', 'a', 'n', 'g', 'u', 'a', 'g', 'e', '\0'};
```

- character pointer (문자 포인터)의 이용 : 상수 형태의 문자열
  - 문자열 변경 불가능

```
char *c4 = "C language";
```



5

character string의 출력

- 문자 배열에서 각 문자를 출력

```
i = 0;
while(c2[i] != '\0')
    printf("%c", c2[i++]);
printf("\n");
```

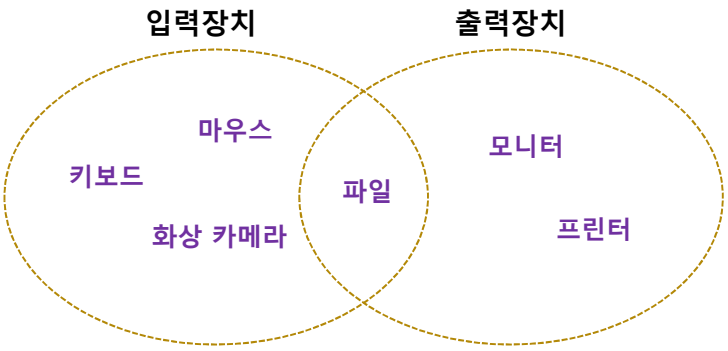
- 문자 포인터의 주소 값을 변경하면서 각 문자를 출력

```
i = 0;
while(*(c4 + i) != '\0')
    printf("%c", *(c4 + i++));
printf("\n");
```

6

Chapter 21-1. 스트림과 데이터의 이동

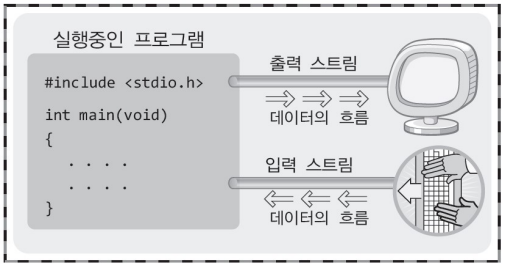
무엇이 입력이고 무엇이 출력인가



- 입출력 장치는 매우 포괄적이다.
- 데이터를 컴퓨터 내부로 받아들이는 것이 입력이고
- 데이터를 외부로 전송하는 것이 출력이다.

8

데이터의 이동수단이 되는 스트림



콘솔 입출력을 위한 스트림은 프로그램이 시작되면 OS에 의해서 자동으로 생성된다.

- **데이터의 입 출력이 가능한 이유!**

  출력의 경로가 되는 출력 스트림과 입력의 경로가 되는 입력 스트림이 존재하기 때문

- **입출력 스트림이란?**

  OS가 데이터의 입출력을 위해 놓아주는 소프트웨어적인 형태의 다리!

9

스트림의 생성과 소멸

• stdin	표준 입력 스트림	키보드 대상으로 입력
• stdout	표준 출력 스트림	모니터 대상으로 출력
• stderr	표준 에러 스트림	모니터 대상으로 출력

- stdin과 stdout은 각각 표준 입력 스트림과 표준 출력 스트림을 의미하는 이름들이다.
- stderr은 표준 에러 스트림이라 하며, 출력의 대상은 stdout과 마찬가지로 모니터이다.
- 출력 리다이렉션이라는 것을 통해서 stdout과 stderr이 향하는 데이터 전송의 방향을 각각 달리 할 수 있다.
- stdin, stdout, stderr은 모두 프로그램 시작과 동시에 자동으로 형성되고 프로그램 종료시 자동으로 소멸된다.
- 이외의 스트림들은 프로그래머가 직접 형성해야 한다. 예를 들어 파일 입출력을 위한 스트림은 직접 형성해야 한다.
  - 스트림이라 불리는 이유는 데이터의 이동을 한 방향으로만 형성하기 때문이다.
  - 물이 한 방향으로 흐르듯 스트림도(스트림은 물의 흐름을 의미함) 한 방향으로만 데이터가 이동한다.

10

Chapter 21-2. 문자 단위 입출력 함수

문자 입출력 함수 (1/2)

- 하나의 문자를 출력하는 두 함수

```
#include <stdio.h>
int putchar(int c);
int fputc(int c, FILE * stream);
    ➜ 함수호출 성공 시 쓰여진 문자정보가, 실패 시 EOF 반환
```

putchar 함수는 인자로 전달된 문자를 모니터에 출력한다.

fputc 함수의 두 번째 인자를 통해서 출력의 대상을 지정한다.

fputc의 두 번째 인자로 stdout이 전달되면 이 putchar 함수와 동일한 결과를 보인다.

예, fputc ('a', stdout);

12

문자 입출력 함수 (2/2)

- 하나의 문자를 입력 받는 두 함수

```
#include <stdio.h>
int getchar(void);
int fgetc(FILE * stream);
    ➡ 파일의 끝에 도달하거나 함수호출 실패 시 EOF 반환
```

키보드에서 입력된 문자의 정보를 반환한다.
문자를 입력 받을 대상정보를 인자로 전달한다.

getchar 함수와 fgetc 함수의 관계는 putchar 함수와 fputc 함수의 관계와 같다.
예, x = fgetc (stdin);

13

문자 입출력 관련 예제

```
int main(void)
{
    int ch1, ch2;

    ch1=getchar();   // 문자 입력
    ch2=fgetc(stdin);   // 엔터 키 입력

    putchar(ch1);   // 문자 출력
    fputc(ch2, stdout);   // 엔터 키 출력
    return 0;
}
```

문자의 입력을 완성하는 엔터 키의
입력도 하나의 문자로 인식이 된다.
따라서 이 역시도 입출력이 가능하다.

ReadWriteChar.c

실행결과

```
p
p
```

첫 번째 P는 입력이 된 P
두 번째 P는 출력된 P

문자를 int형 변수에 저장하는 이유는 EOF를 설명하면서 함께 설명한다.

14

문자 입출력에서의 EOF

- **EOF의 의미**
  - ▶ EOF는 End Of File의 약자로서, 파일의 끝을 표현하기 위해서 정의해 놓은 상수이다.
  - ▶ 파일을 대상으로 fgetc 함수가 호출되었을 때 파일에 끝에 도달하면 EOF가 반환된다.
- **콘솔 대상의 fgetc, getchar 함수호출로 EOF (-1)를 반환하는 경우**
  - ▶ 함수호출의 실패
  - ▶ Windows에서 Ctrl+Z 키, Linux에서 Ctrl+D 키가 입력이 되는 경우

ConsoleEOF.c

```
int main(void)
{
    int ch;

    while(1)
    {
        ch=getchar();
        if(ch==EOF)
            break;
        putchar(ch);
    }
    return 0;
}
```

실행결과

```
Hi~
Hi~
I like C lang.
I like C lang.
^Z
```

- 키보드에는 EOF가 존재하지 않는다.
- 따라서 EOF를 Ctrl+Z 키 또는 Ctrl+D 키로 약속해 놓은 것이다.
- 예제에서 보이듯이. 하나의 문장이 입력되어도 문장을 이루는 모든 문자들이 반복된 getchar 함수의 호출을 통해서 입력될 수 있다.

15

반환형이 int이고, int형 변수에 문자를 담는 이유는?

```
int getchar(void);
int fgetc(FILE * stream);
```

- **반환형이 char형이 아닌 int형인 이유는?**
  - ▶ char형은 예외적으로 signed char가 아닌 unsiged char로 표현하는 컴파일러가 존재한다.
  - ▶ 파일의 끝에 도달했을 때 반환하는 EOF는 -1로 정의되어 있다.
  - ▶ char를 unsigend char로 표현하는 컴파일러는 EOF에 해당하는 -1을 반환하지 못한다.
  - ▶ int는 모든 컴파일러가 signed int로 처리한다. 따라서 -1의 반환에 무리가 없다.
  - ▶ Visual Studio에서는 char로 사용해도 무방하다.

16

실습문제

- getchar 및 putchar를 이용한 문자열 입출력 프로그램
  - getchar 함수를 연속으로 사용하여 영어 문장을 입력 받음
    - !가 입력 되면 문장의 끝이라는 표시이고 !는 문장에 포함시키지 않음
    - 입력된 문장을 printf 함수를 이용하여 출력함
    - 동일한 문장을 putchar 함수를 이용하여 출력함
    - 출력 결과는 아래와 같도록 함

```
영어 문장을 입력 : Information and Communications!
printf로 출력한 결과 : Information and Communications
putchar로 출력한 결과 : Information and Communications
Press any key to continue.
```

17

실습문제 - program

```
int main(void) {
    int i=0; char ch[80];
    while(1) {
        ch[i] = getchar();
        if(ch[i] == '!') break;
            i++;
    }
    ch[i]='\0';
    printf("printf ch = %s\n", ch);
    printf("putchar ch = ");
    i=0;
    while(ch[i]!='\0') {
        putchar(ch[i]);
        i++;
    }
    printf("\n");
    return 0;
}
```

18

Chapter 21-3. 문자열 단위 입출력 함수

문자열 출력 함수: puts, fputs

```
#include <stdio.h>
int puts(const char * s);
int fputs(const char * s, FILE * stream);
```

➜ 성공 시 0이 아닌 값을, 실패 시 EOF 반환

인자로 전달되는 문자열을 출력한다. 단 fputs 함수는 두 번째 인자를 통해서 출력의 대상을 지정(키보드, 파일) 할 수 있다.

WriteString.c

```
int main(void)
{
    char * str="Simple String";
    printf("1. puts test ------ \n");
    puts(str);
    puts("So Simple String");
    printf("2. fputs test ----- \n");
    fputs(str, stdout); printf("\n");
    fputs("So Simple String", stdout); printf("\n");
    printf("3. end of main ----\n");
    return 0;
}
```

puts 함수가 호출되면 문자열 출력 후 자동으로 개행이 이뤄지지만, fputs 함수가 호출되면 문자열 출력 후 자동으로 개행이 이뤄지지 않는다는 사실에 주목!

실행결과

```
1. puts test ------
Simple String
So Simple String
2. fputs test -----
Simple String
So Simple String
3. end of main ----
```

20

문자열 입력 함수: gets, fgets

```
#include <stdio.h>
char * gets(char * s);
char * fgets(char * s, int n, FILE * stream);
```

➜ 파일의 끝에 도달하거나 함수호출 실패 시 NULL 포인터 반환

```
int main(void)
{
    char str[7];   // 7바이트의 메모리 공간 할당
    gets(str);   // 입력 받은 문자열을 배열 str에 저장
    . . . .
}
```

이 경우 입력되는 문자열의 길이가 배열을 넘어설 경우 할당 받지 않은 메모리를 참조하는 오류가 발생한다.

```
int main(void)
{
    char str[7];
    fgets(str, sizeof(str), stdin);
    . . . .  // stdin으로부터 문자열 입력 받아서 str에 저장
}
```

stdin으로부터 문자열을 입력 받아서 str에 저장을 하되, 널 문자를 포함하여 sizeof(str)의 크기만큼 저장을 해라.

21

fgets 함수의 호출의 예

ReadString.c

```
int main(void)
{
    char str[7];
    int i;

    for(i=0; i<3; i++)
    {
        fgets(str, sizeof(str), stdin);
        printf("Read %d: %s \n", i+1, str);
    }
    return 0;
}
```

실행결과1

```
12345678901234567890
Read 1: 123456
Read 2: 789012
Read 3: 345678
```

6개의 문자씩 끊어서 읽히고 있다.
즉, 한번의 fgets 함수호출당 최대 6개의 문자만 읽혀진다. (배열 str의 방은 7개)

실행결과2

```
We 엔터
Read 1: We

like 엔터
Read 2: like

you 엔터
Read 3: you
```

Enter 키 (ASCII 값 10)의 입력도 문자열의 일부로 받아 들임을 보임

실행결과3

```
Y & I 엔터
Read 1: Y & I

ha ha 엔터
Read 2: ha ha

^^ -- 엔터
Read 3: ^^ --
```

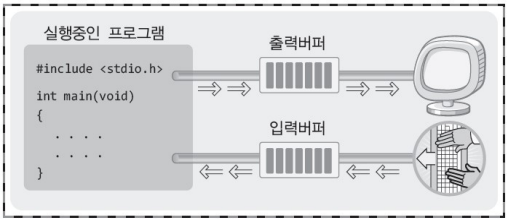
공백을 포함하는 문자열을 읽어 들임을 보임

22

Chapter 21-4. 표준 입출력 버퍼

표준 입출력 기반의 버퍼와 버퍼링의 이유

- **입출력 버퍼**
  - 버퍼는 특정 크기의 메모리 공간을 의미한다.
  - 운영체제는 입력과 출력을 돕는 입출력 버퍼를 생성하여 제공한다.
  - 표준 입출력 함수를 기반으로 데이터 입출력 시 입출력 버퍼를 거친다.
- **입출력 버퍼에 데이터가 전송되는 시점**
  - 호출된 출력함수가 반환이 되는 시점이 출력버퍼로 데이터가 완전히 전송된 시점이다.
  - 엔터를 입력하는 시점이 키보드로 입력된 데이터가 입력버퍼로 전달되는 시점이다.



버퍼링을 하는 이유는 데이터 이동의 효율과 관련이 있다. 데이터를 모아서 전송하면, 하나씩 전송하는 것보다 효율적이다.

24

출력버퍼를 비우는 fflush 함수

```
#include <stdio.h>
int fflush(FILE * stream);
```

➜ 함수호출 성공 시 0, 실패 시 EOF 반환

- **인자에 해당하는 출력버퍼를 비운다.**

  출력버퍼를 비운다는 것은 출력버퍼에 저장된 데이터를 지우는 것이 아니라, 출력버퍼에 저장된 데이터를 목적지로 최종 전송함을 뜻한다.

- **fflush(stdout) → 출력버퍼를 비워라!**

출력버퍼의 경우와 달리 입력버퍼의 비움은 입력버퍼에 저장된 데이터의 소멸을 뜻한다.
그리고 fflush 함수는 출력버퍼를 대상으로 정의된 함수이다. 따라서 fflush(stdin) 과 같은 형태의 함수호출은 그 결과를 보장받지 못한다.

그렇다면 입력버퍼는 어떻게 비워야 할까?

25

입력버퍼는 어떻게 비워야 하나요?

주민번호 앞 6자리만 입력 받기 위해서 배열의 길이가 널 문자 포함 7이다.

NeedInputBufFlush.c

```
int main(void)
{
    char perID[7];
    char name[10];

    fputs("주민번호 앞 6자리 입력: ", stdout);
    fgets(perID, sizeof(perID), stdin);

    fputs("이름 입력: ", stdout);
    fgets(name, sizeof(name), stdin);

    printf("주민번호: %s \n", perID);
    printf("이름: %s \n", name);
    return 0;
}
```

실행결과1

```
주민번호 앞 6자리 입력: 950915
이름 입력: 주민번호: 950915
이름:
```

엔터 키가 남아서 문제가 되는 상황

실행결과2

```
주민번호 앞 6자리 입력: 950709-1122345
이름 입력: 주민번호: 950709
이름: -1122345
```

말 안 듣는 사람들 때문에 문제되는 상황

26

입력버퍼는 어떻게 비워야 하나요?

```
void ClearLineFromReadBuffer(void)
{
    while(getchar()!='\n');
}
```

입력버퍼를 비우는 함수

어떠한 경우에도 주민번호 6자리만 입력 받도록 재 구현된 예제

```
int main(void)
{
    char perID[7];
    char name[10];

    fputs("주민번호 앞 6자리 입력: ", stdout);
    fgets(perID, sizeof(perID), stdin);
    ClearLineFromReadBuffer();   // 입력버퍼 비우기

    fputs("이름 입력: ", stdout);
    fgets(name, sizeof(name), stdin);

    printf("주민번호: %s\n", perID);
    printf("이름: %s\n", name);
    return 0;
}
```

InputBufFlush.c

27

Chapter 21-5. 입출력 이외의 문자열 관련 함수

문자열의 길이를 반환하는 함수: strlen

```
#include <string.h>
size_t strlen(const char * s);
```

➜ 전달된 문자열의 길이를 반환하되, 널 문자는 길이에 포함하지 않는다.

size_t의 일반적인 선언
typedef unsigned int size_t;
typedef에 관해서는 뒤에 설명

```
int main(void)
{
    char str[]="1234567";
    printf("%u \n", strlen(str));
    . . . .     // 문자열의 길이 7이 출력
}
```

실행결과

```
문자열 입력: Good morning
길이: 13, 내용: Good morning

길이: 12, 내용: Good morning
```

```
void RemoveBSN(char str[])
{
    int len=strlen(str);
    str[len-1]=0;
}

int main(void)
{
    char str[100];
    printf("문자열 입력: ");
    fgets(str, sizeof(str), stdin);
    printf("길이: %d, 내용: %s \n", strlen(str), str);

    RemoveBSN(str);
    printf("길이: %d, 내용: %s \n", strlen(str), str);
    return 0;
}
```

마지막에 삽입되는
Enter 키 (ASCII 숫자 10)
문자를 없애는 예제

RemoveBSN.c

29

문자열을 복사하는 함수들: strcpy, strncpy

```
#include <string.h>
char * strcpy(char * dest, const char * src);
char * strncpy(char * dest, const char * src, size_t n);
```

➜ 복사된 문자열의 주소 값 반환

대표적인 문자열
복사 함수

```
int main(void)
{
    char str1[30]="Simple String";
    char str2[30];
    strcpy(str2, str1);
    . . . .    // str1의 문자열을 str2에 복사
}
```

str1에 저장된 문자열을 str2에 단순히 복사!
strcpy 함수를 호출하는 경우 배열의
범위를 넘어서 복사가 진행될 위험이 있다.

```
int main(void)
{
    char str1[30]="Simple String";
    char str2[30];
    strncpy(str2, str1, sizeof(str2));
    . . . .
}
```

str1에 저장된 문자열을 str2에 복사하되
최대 sizeof(str2)의 반환 값 크기만큼
복사한다.

30

strncpy 함수를 잘못 사용한 예

StringCopyCase.c

```
int main(void)
{
    char str1[20]="1234567890";
    char str2[20];
    char str3[5];

    /**** case 1 ****/
    strcpy(str2, str1);
    puts(str2);

    /**** case 2 ****/
    strncpy(str3, str1, sizeof(str3));
    puts(str3);

    /**** case 3 ****/
    strncpy(str3, str1, sizeof(str3)-1);
    str3[sizeof(str3)-1]=0;
    puts(str3);
    return 0;
}
```

실행결과

```
1234567890
12345傲傲傲傲傲?234567890
1234
```

배열 길이 str1에 딱 맞는 길이만큼만 복사를 하겠다는 의도의 문장

두 번째 strncpy 함수호출 후의 결과에 이상이 보이는 이유는 복사하는 과정에서 문자열의 끝을 의미하는 널 문자가 복사되지 않았기 때문이다.
문자열을 복사할 때에는 항상 널 문자의 복사까지 고려해야 한다.

31

문자열을 덧붙이는 함수들: strcat, strncat

```
#include <string.h>
char * strcat(char * dest, const char * src);
char * strncat(char * dest, const char * src, size_t n);
    ➡ 덧붙여진 문자열의 주소 값 반환
```

strncat 함수는 덧붙일 문자열의 최대 길이를 제한한다.

최대 n개의 문자를 덧붙이되 널 문자 포함하여 n+1개의 문자를 덧붙인다.

```
int main(void)
{
    char str1[30]="First~";
    char str2[30]="Second";
    strcat(str1, str2);
    . . . .  // str1의 문자열 뒤에 str2를 복사
}
```

덧붙여지는 방식

StringConcatCase.c

```
int main(void)
{
    char str1[20]="First~";
    char str2[20]="Second";
    char str3[20]="Simple num: ";
    char str4[20]="1234567890";

    /**** case 1 ****/
    strcat(str1, str2);
    puts(str1);

    /**** case 2 ****/
    strncat(str3, str4, 7);
    puts(str3);
    return 0;
}
```

실행결과

```
First~Second
Simple num: 1234567
```

32

문자열을 비교하는 함수들: strcmp, strncmp

```
#include <string.h>
int strcmp(const char * s1, const char * s2);
int strncmp(const char * s1, const char * s2, size_t n);
    ➡ 두 문자열의 내용이 같으면 0, 같지 않으면 0이 아닌 값 반환
```

- s1이 더 크면 0보다 큰 값 반환
- s2가 더 크면 0보다 작은 값 반환
- s1과 s2의 내용이 모두 같으면 0 반환

strncmp는 최대 n개의 문자를 앞에서부터 비교

- **크고 작음은 아스키코드 값을 근거로 한다.**
- **A보다 B가, B보다 C가 아스키 코드 값이 더 크고 A보다 a가 , B보다 b가 아스키 코드 값이 더 크니, 사전편찬순서를 기준으로 뒤에 위치할 수록 더 큰 문자열로 인식해도 된다.**

```
printf("%d", strcmp("ABCD", "ABCC"));
printf("%d", strcmp("ABCD", "ABCDE"));
```

0보다 큰 값이 출력

0보다 작은 값이 출력

두 문자열이 같으면 0, 다르면 0이 아닌 값을 반환한다고 인식하고 있어도 충분하다!

33

문자열 비교의 예

StringCompCase.c

```
int main(void)
{
    char str1[20];
    char str2[20];
    printf("문자열 입력 1: ");
    scanf("%s", str1);
    printf("문자열 입력 2: ");
    scanf("%s", str2);

    if(!strcmp(str1, str2))
    {
        puts("두 문자열은 완벽히 동일합니다.");
    }
    else
    {
        puts("두 문자열은 동일하지 않습니다.");

        if(!strncmp(str1, str2, 3))
            puts("그러나 앞 세 글자는 동일합니다.");
    }
    return 0;
}
```

실행결과

```
문자열 입력 1: Simple
문자열 입력 2: Simon
두 문자열은 동일하지 않습니다.
그러나 앞 세 글자는 동일합니다.
```

34

그 이외의 변환함수들

```
int atoi(const char * str);        문자열의 내용을 int형으로 변환
long atol(const char * str);       문자열의 내용을 long형으로 변환
double atof(const char * str);     문자열의 내용을 double형으로 변환
```

헤더파일 stdlib.h에 선언

ConvStringToPrimitive.c

```
int main(void)
{
    char str[20];
    printf("정수 입력: ");
    scanf("%s", str);
    printf("%d \n", atoi(str));
    printf("실수 입력: ");
    scanf("%s", str);
    printf("%g \n", atof(str));
    return 0;
}
```

위의 함수들을 모른다면 문자열에 저장된 숫자 정보를 int형 또는 double형으로 변환하는 일은 번거로운 일이 될 수 있다.

실행결과

```
정수 입력: 15
15
실수 입력: 12.456
12.456
```

35

실습문제 (Lab 1)

- 적당한 길이의 문자열을 입력 받아서 그 안에 존재하는 숫자의 총 합을 계산해서 출력하는 프로그램을 작성하라.
  - (ex) 사용자로부터 입력 받은 문자열이 "Z19#56$"이면 연산 결과는 1+9+5+6 = 21이 출력되어야 함
  - atoi() 함수를 사용할 것 (#include <stdlib.h>)
  - 문자열을 저장할 수 있는 충분한 크기의 문자열 배열 선언

36

실습 시간 (2018년 10월 29일)

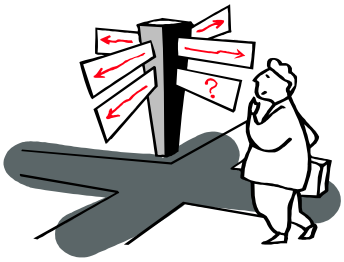
- 예제 (21장): ConsoleEOF.c, ConStringToPrimitive.c, InputBufFlush.c, NeedInputBufFlush.c, ReadString.c, ReadWriteChar.c, RemoveBSN.c, StringCompCase.c, StringConCatCase.c, StringCopyCase.c, WriteString.c (11개)
- Lab 문제: Lab1.c

37

고민: 왜 그럴까??

- 미션: 숙박관리 프로그램 (version 1) 에서 사용자 이름에 한 글자가 아닌 이름을 저장하기
  - (1) char room[3][5] ➔ char *room[3][5]
  - (2) 초기화: room[i][j] = '\0' ➔ room[i][j] = "\0"
  - (3) 새로운 변수 선언하여 이름 입력하기:
    - char name[30]
    - printf("고객이름(name):")
    - scanf("%s", name)
    - room[i][j] = name;

38



문자열 관련 함수들의 사용법을 배운
Chapter 21이 끝났습니다. 질문 있으신지요?