1 장 언어소개

1-1. hello world 출력

> 실습 전 이론

01. C 프로그램은 항상 하나 이상의 함수로 구성된다 **.**

02. 함수의 이름은 자유지만 주 프로그램의 이름은 꼭 **main**으로 한다 **.**

03. 이번 프로그램에서 사용한 함수는 **printf**이며 **stdio(**표준 입출력 라이브러리 **)**에 들어있다 **.**

03-1 #include <stdio.h>를 통해 라이브러리를 불러온다 **.**

04. 함수들 사이에 데이터가 오갈땐 매개변수를 사용한다 **.**

04-1. 매개변수는 함수 뒤의 괄호에 써준다 **. ex. main(params)**

04-2. 이번 프로그램에선 매개변수가 존재하지 않으며 **,** 매개변수가 없더라도 괄호는 꼭 존재하여야 한다 **.**

05. 중괄호 **{}** 는 문장들을 묶어서 하나의 덩어리로 만들어주는 역할을 한다

05-1. 이번 프로그램에선 **{** 와 **}** 의 사이가 **main** 함수의 내용임을 나타내고 있다 **.**

> **Hello world** 출력

모든 언어의 시작이 아닐까

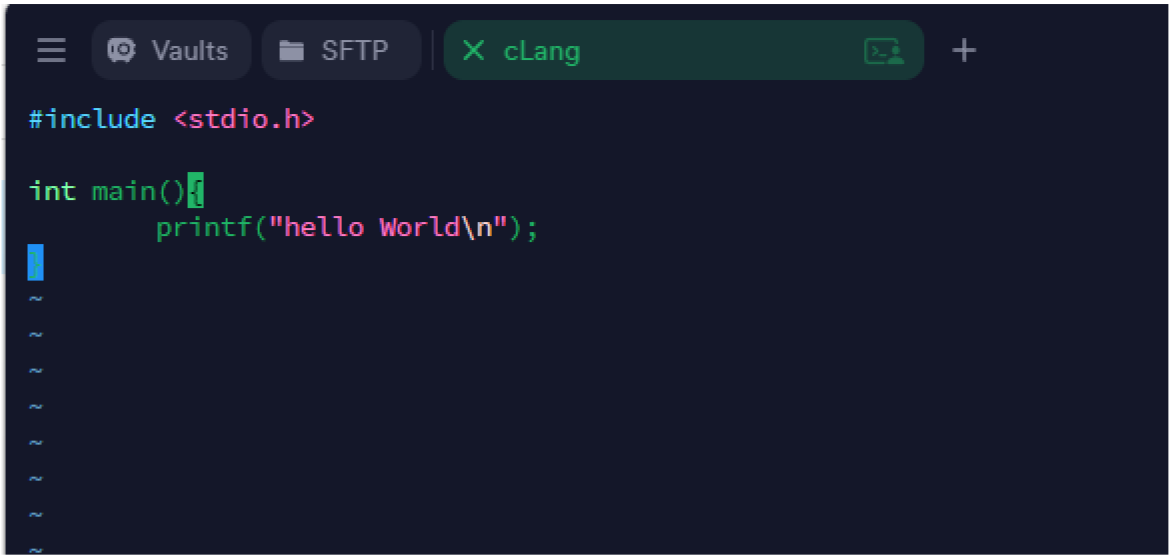
개발 환경은 **Ubuntu**를 설치했다 **.**

```
sudo apt-get update
```

위 명령어를 통해 업데이트를 진행하고

```
vi hello.c
```

위 명령어를 통해 **c** 언어 파일을 생성한다 **.**

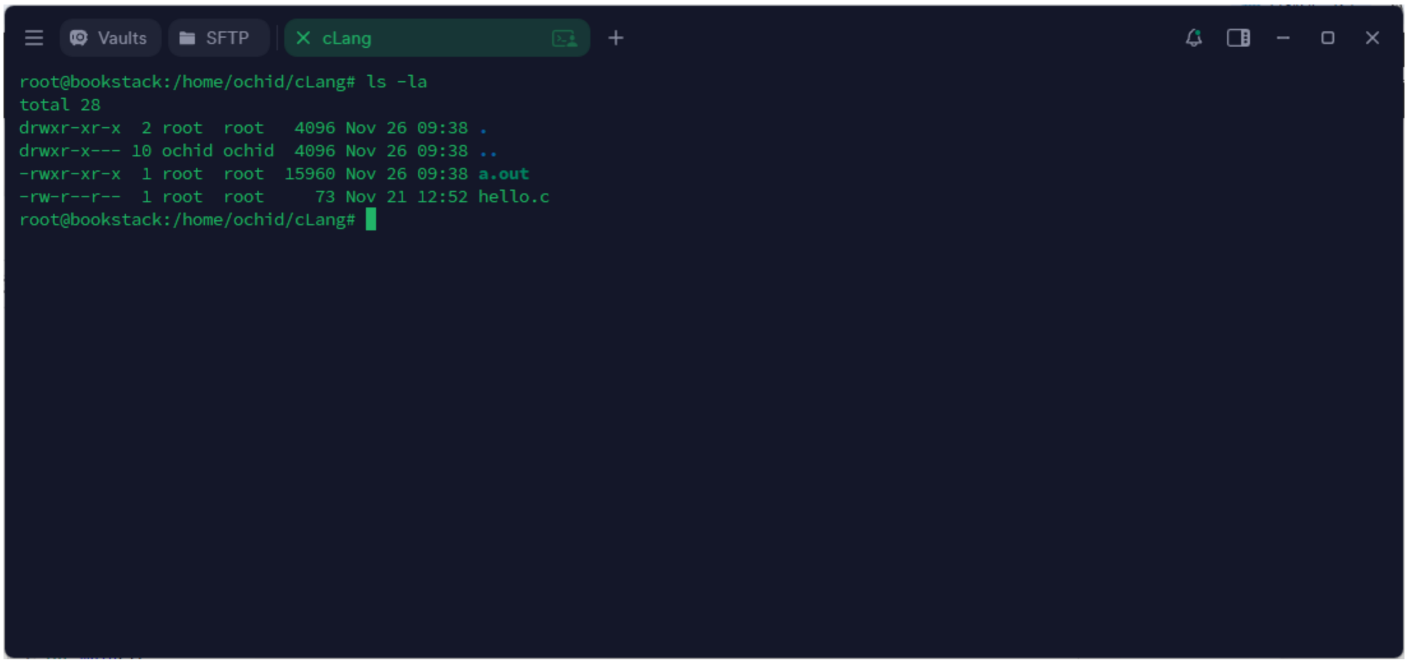



```
#include <stdio.h>

int main(){
  printf("hello World\n");
  return 1;
}
```

위 명령어를 삽입 후

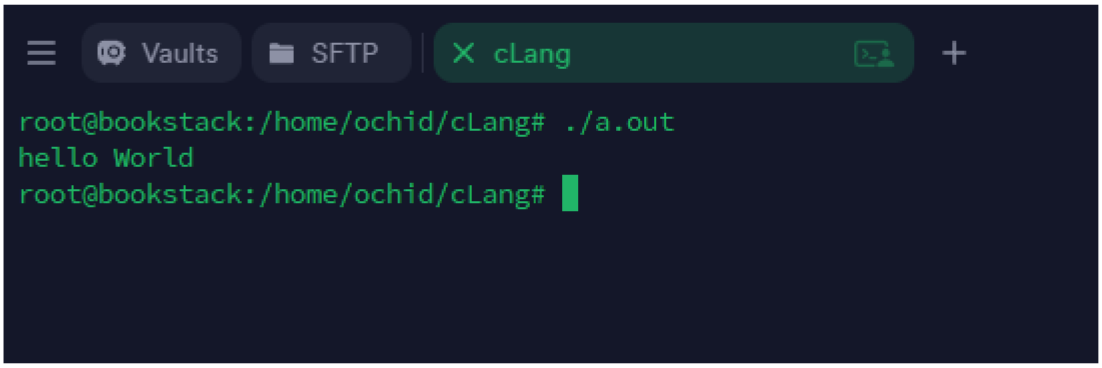
```
cc hello.c
```

명령어를 사용하면 컴파일이 진행된다 .



따로 이름을 지정하지 않았으므로 **a.out** 이라는 파일이 생성되었다 .

*파이썬은 따로 컴파일 과정이 필요없었는데 언어 특성상이지만 역체감이 심한듯 .




```
./a.out
```

명령어를 통해 현재 디렉터리의 **a.out** 파일을 실행하면 정상적으로 **hello world**가 출력됨을 확인할 수 있다 .

> 실습 후 이론

01. printf("hello, world\n"); 라는 문장은 **prinf** 라는 함수를 호출하여 매개변수에 **"hello, world\n"** 를 포함하고있다 .

02. prinf 함수는 화면출력을 담당하는 함수이다 .

03. \n 은 개행으로 쓰인다 . *파이썬이랑 같네 몰랐다 .

03-1. 같은 원리로 **\t** 은 **tab**키 **, \b** 는 백스페이스 역할을 한다 **. (\b**는 실제로 문자열 하나가 지워진다 **.)**

1-2. 변수와 수식 표현

실습 전 이론

01. 이번 챕터에선 변수와 수식을 통해 화씨 온도를 섭씨 온도로 바꿔 출력하는 프로그램을 만들어본다 .

02. 코드를 그대로 복사 , 붙여넣지 말자 , 결국 같은 코드를 가져다 쓴다해도 직접쓰는게 배울땐 훨씬 좋다 . 복붙은 휘발된다 .

03. 코드가 이해가 안되면 넘어가지 마라 , 번역을 한다고 생각하고 한줄한줄 무슨 과정이 이어지는지 생각하면서 봐야한다 .

04. 이번 프로그램을 통해 변수 지정 및 반복문을 배워볼 수 있다 .

05. //** 를 통해 주석을 사용하여 설명을 넣어놓을 수 있다 .

05-1. /* 예시를 들면 이런식으로 주석이 들어간다 **.*/**

06. C에서 모든 변수는 사용되기 전에 선언해 주어야한다 .

07. 변수 지정 시 큰따옴표만 사용가능하다 **. (**맙소사 **)**

08. printf() 에 변수를 포함할 경우 무조건적으로 서식 지정자를 사용하여야 한다 .

123 이라는 정수 값이 지정된 **num** 변수를 **printf()**에서 출력하려면

파이썬처럼 **print(num)**으로 사용하면 안되고 **, printf("%d",num)** 같은 방식으로 변수 값을 불러와줘야한다 .

찾아보니 **printf**는 메모리에 있는 값만 가져올 뿐 가져오는 항목의 타입을 알 수 없어 서식 지정자를 통해 이를 알려줘야한다고 한다 .

*이또한 파이썬에서 온 역체감

실습

```
#include <stdio.h>

int main()
{
```

```
        int fahr, celsius;
        int lower, upper, step;

        lower = 0;
        upper = 300;
        step = 20;

        fahr = lower;
        while (fahr <=upper) {
        celsius = 5 * (fahr-32) / 9;
        printf("%d\t%d\n", fahr, celsius);
        fahr = fahr + step;
        }
        return 1;
}
```

파이썬을 쓸 줄 알아서 그런가 아직까지는 쉽다 .

위 이론에서 **C** 언어의 모든 변수는 사용 전 선언을 해야한다고 적어뒀듯

```
int fahr, celsius;
        int lower, upper, step;

        lower = 0;
        upper = 300;
        step = 20;
```

위와 같이 변수들을 선언한 후 변수에 값을 넣어준다 .

쉽게 설명하면

int lower; -> 나는 **lower** 이라는 변수에 숫자 **(int)**를 지정할꺼다 **! (**정확히 말하자면 **int**는 정수형이다 **.)**

lower = 300; -> 아 이전에 **lower**에는 **int**를 넣는다고 했으니 **300**은 문자열 **300**이 아닌 정수 **300** 이겠구나 .

라고 설명할 수 있을 것 같다 .

while (fahr <=upper){

...

}

는 **fahr** 값이 **upper** 보다 작거나 같을 경우 **while()**내부의 코드가 계속 반복된다 .

매 코드가 반복될때마다 **fahr** 변수에 **step** 만큼이 추가되어 다시 반복문이 실행되다가 **upper** 보가 커지면 반복문이 종료되는 구조라고 볼 수 있다 .

코드를 실행해보면

```
root@bookstack:/home/ochid/cLang# ./a.out
0       -17
20      -6
40      4
60      15
80      26
100     37
120     48
140     60
160     71
180     82
200     93
220     104
240     115
260     126
280     137
300     148
root@bookstack:/home/ochid/cLang#
```

왼쪽에는 **fahr** 에서 **step**의 값인 **20** 씩 올라가고 오른쪽엔 해당 **fahr**을 **celsius**로 변환하여 출력이 되고 있다 .

```
printf("%3d\t%3d\n", fahr, celsius);
```

위 코드 처럼 **%d**를 **%3d**로 바꿔주면

```
root@bookstack:/home/ochid/cLang# ./a.out
  0     -17
 20      -6
 40       4
 60      15
 80      26
100      37
120      48
140      60
160      71
180      82
200      93
220     104
240     115
260     126
280     137
300     148
```

이런식으로 조금 보기 편하게 문자열이 정렬되는 것을 확인할 수 있다 . 최대 값이 세자리기 때문에 **3** 을 넣어준 것

```
#include <stdio.h>

int main()
{
        float fahr, celsius;
        int lower, upper, step;

        lower = 0;
        upper = 300;
        step = 20;

        fahr = lower;
        while (fahr <=upper) {
        celsius = 5.0 * (fahr-32.0) / 9.0;
        printf("%3.0f\t%6.1f\n", fahr, celsius);
        fahr = fahr + step;
        }
        return 1;
}
```

변수를 지정할때 **int**가 아닌 **float**으로 변경되었고 서식 지정자를 사용할때 **%d** 에서 **%f** 로 바뀌었다 .

%6.1f 는 전체 **6**자리 , 소수점 **1**자리의 부동소수점 형식으로 값이 출력되도록 지정한다 .

*같은 원리로 **%3.0f** 는 세자리숫자 , 소수점 자리는 보여주지 않는 방식이다

5.0, 32.0 과 같이 실수로 쓰지 않아도 계산은 되지만 식별을 위해 소수점까지 작성하였다 .

```
root@bookstack:/home/ochid/cLang# ./a.out
  0      -17.8
 20       -6.7
 40        4.4
 60       15.6
 80       26.7
100       37.8
120       48.9
140       60.0
160       71.1
180       82.2
200       93.3
220      104.4
240      115.6
260      126.7
280      137.8
300      148.9
```

> 실습 후 이론

01. int는 정수형을 뜻하는 것처럼 다양한 변수타입이 존재한다 . 다음과 같다 .

char 문자형 , **1**바이트

> * **char A** 로 **A**를 지정할 경우 **A** 에는 **1**바이트의 문자만 지정할 수 있다 .
>
> ** 그러므로 **ABCD** 같은 **1**바이트 이상의 문자를 지정할 경우 **char A[] = "ABCD"** 같은 형식을 통해 변수에 지정을 해주어야한다 .

short 단정도 정수형

long 배정도 정수형

float 부동소수점형

double 배정도 부동소수점형

등이 있다 .

*설명과 함께 정리를 해두려했는데 생각보다 관련내용이 많아 추후에 설명하기로 ;

02. 온도 변환 공식은 다음과 같다 **.**

celsius = 5 * (fahr-32) / 9;

위 공식은

celsius = (5/9) * (fahr-32);

로 표현할 수도 있지만 **C** 언어상에서 정수끼리의 계산 결과는 무조건 정수로 출력된다 **. (** 소수점이 없음 **)**

그러므로 **5/9** 는 **0** 이되고 **fahr**에 어떤 값이 들어가도 **celsius**는 항상 **0**인 결과가 나온다 **.**

이런 특이사항도 있을 수 있다는 걸 알아두면 좋을듯

1-3. For 문

> 실습 전 이론

01. 이전 챕터에서 온도변환 프로그램을 작성했었다 . 해당 프로그램에서는 반복을 위해 **while**을 사용했었고 **for** 로변환하여 사용해본다 .

> 실습

```
#include <stdio.h>


int main()
{
        int fahr;

        for (fahr = 0; fahr <=300; fahr = fahr + 20)
                printf("%3d %6.1f\n", fahr, (5.0/9.0) * (fahr-32) );
}
```

이전 챕터에서 생성한 온도변환프로그램을 **while** 문에서 **for**문으로 변환하였다 .

fahr 변수만 남겨두었고 **printf** 부분에 변수가 아닌 계산하는 수식을 넣어서 출력하는 부분도 흥미롭다 .

for문 뒤의 괄호를 **;** 로 분리해서 설명을 해보면

fahr 를 먼저 **0**으로 지정하고 ,

fahr 이 **300** 보다 작거나 같은경우 **for**문이 실행된다 .

마지막으로 **fahr**에 **20**이 더해지고 **for**문이 실행되는 조건이 만족되는지 검사된다 .

> 실습 후 이론

갑자기 궁금해서 **C**와 파이썬의 속도를 비교해봤다 .

단순히 **0**부터 **10000000(**천만 **)** 까지 **print** 하는 코드를 짜서 실행시켰다 .

```
9999994
9999995
9999996
9999997
9999998
9999999
10000000

[Done] exited with code=0 in 9.479 seconds
```

C언어는 **9.4**초

```
9999992
9999993
9999994
9999995
9999996
9999997
9999998
9999999
10000000

[Done] exited with code=0 in 42.517 seconds
```

파이썬은 **42.5** 초

생각보다 엄청 차이가 많이나네 **...** 간단한 숫자세기도 이런데 프로그램이 무거워질수록 차이가 커질듯하다 .

물론 내가 코드를 잘 짜야겠지만

1-4. 상수 정의

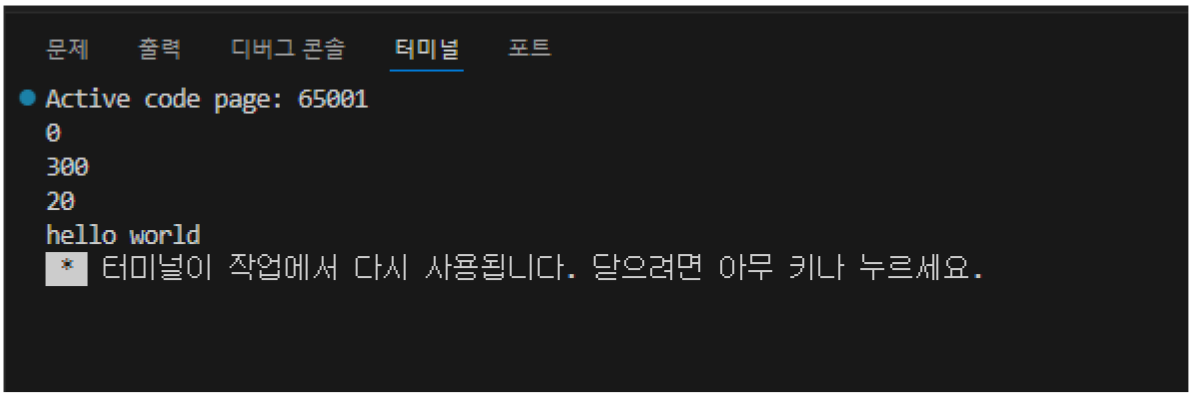
실습 전 이론

이전 프로그램에서 변수에 **300,200** 같은 정수를 지정했었다 .

하지만 변수는 말그대로 변할 수 있는 값 , 코드 중간에 어디서 변화가 생길지 모르기 때문에 변할일이 없는 값은 상수로 정의를 한다 .

실습

```
#include <stdio.h>

#define LOWER 0
#define UPPER 300
#define STEP 20
#define TEXT "hello world"

int main()
  {
    printf("%d\n", LOWER);
    printf("%d\n", UPPER);
    printf("%d\n", STEP);
    printf("%s\n", TEXT);
    return 0;
  }
```



정상적으로 상수가 정의되고 출력됨을 확인

*아 리눅스에서 하나하나 컴파일하고 실행시키기 귀찮아서 **VSCode**로 넘어왔다 .

https://velog.io/@jamkris/Visual-Studio-Code%EC%97%90%EC%84%9C-C%EC%96%B8%EC%96%B4-%EC%BB%B4%ED%8C%8C%EC%9D%BC-%ED%95%98%EA%B8%B0

vscode C 컴파일 세팅은 위 링크를 참고했다 .

> 실습 후 이론

보통 코드의 종료시점은 **;** 로 구분되는데 상수를 지정할 경우 **;** 를 쓰지 않는게 재밌었다 .

1.5.1 문자 입출력

> 실습 전 이론

프로그램하면 빠질 수 없는게 또 사용자의 입력 값을 받아서 처리하는 부분이다 .

파이썬을 공부할때도 **input()**을 알게 되면서 부터 재미를 확 느꼈던 기억이 있다 .

이번 실습에서는 입력 값을 받는 함수 **getchar()** 와

문자 하나를 출력하는 함수 **putchar()**를 사용한다 .

이번 프로그램에선 **EOF** 라는 개념이 등장한다 . **End of File** 의 줄임말이며 , 파일의 끝을 나타낸다 .

프로그램의 진행은 다음과 같다 .

1. 문자를 읽어 들인다 .

2. while

3. 문자를 출력한다 .

4. 다음 문자를 읽어들인다 .

> 실습

```
#include <stdio.h>


int main(){
  int c;

  c = getchar();
  while (c != EOF) {
```

```
    putchar(c);
    c = getchar();
    }
  return 0;
  }
```

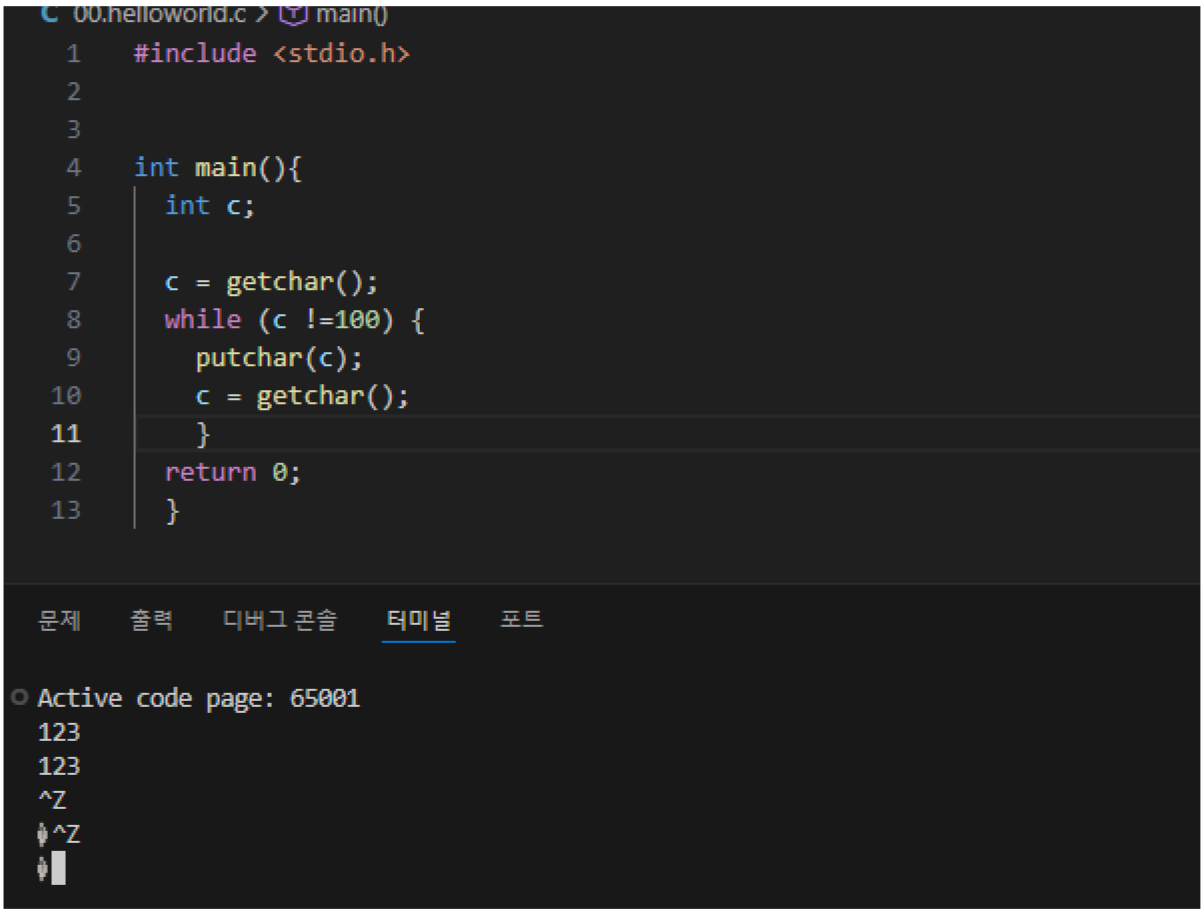
c 를 선언 , **while**문이 실행되기 전 **getchar**를 통해 입력 값을 받고

받은 입력 값이 **EOF** 가 아닌지 먼저 확인한다 .

EOF 는 보통 **-1** 아니면 **0** 으로 되어있으며 ,

그렇다고 우리가 입력에 **-1** 을 넣는다고 실행되는 것이 아니다 .

EOF 를 넣으려면 **control + z** 를 누른 상태로 **enter** 키를 눌러주면 된다 .



c 가 **EOF**인지 확인하는 부분을 제거할 경우 **control + z + Enter** 키에도 아무 반응이 없는 것을 확인할 수 있다 .

아무튼 프로그램 자체는 사용자의 입력 값을 그대로 출력하는 것을 확인 할 수 있다 .

putchar() 는 기본적으로 개행이 들어간다는 것도 흥미로운듯

1.5.2 문자의 갯수 세기

> 실습 전 이론

이전에 만들었던 입력 **값 복사** 프로그램에서 조금 변형된 버전이다 .

이전에는 그냥 들어온 값을 그대로 출력했다면

이번에는 들어온 값의 문자 갯수를 세는 프로그램으로 만들어보자 **ex. abcd -> 4**

> 실습

```
#include <stdio.h>



int main(){
  int c;
  int count;

  c = getchar();
  count = 0;

  while(c != EOF){
    ++count;
    c = getchar();
  }
  printf("%d\n", count);
  return 0;
  }
```

위 코드는 내가 생각해서 만든거고 아래가 책에 나와있는 정석 코드다 .

```
#include <stdio.h>

int main(){
  long nc;
  nc=0;
```

```
  while (getchar () != EOF)
    ++nc;
  printf("%1d\n", nc);
  return 0;
}
```

getchar를 굳이 변수로 지정하지 않아서 좀 더 간결해졌다 .

long은 **16**바이트 만 받을 수 있는 **int** 보다 더 높은 값도 지정할 수 있는 타입이다 .

> 실습 후 이론

1.5.3 행의 갯수 세기

> 실습 전 이론

입력 받은 문자의 개행 갯수를 찾기 위해선 **\n** 기호를 찾으면 된다 .

"안녕 **\n**하세요 **."** 를 출력하면

안녕

하세요

로 출력이 된다는 말이다 .

이번 프로그램에선 **if** 문도 사용하게 된다 .

> 실습

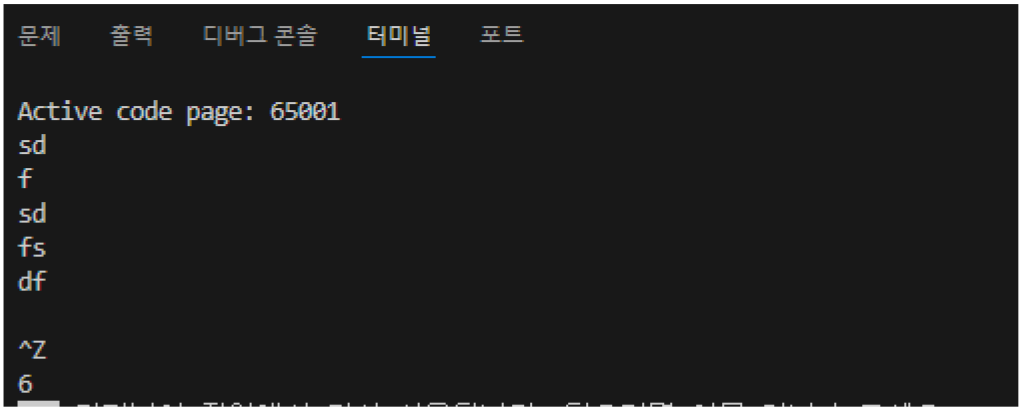
```
#include <stdio.h>

int main()
  {
  int c, nl;

  nl = 0;
  while ((c = getchar())!=EOF)
    if (c =='\n')
      ++nl;
    printf("%d\n", nl);
  }
```

위 코드에서 보면 **while** 문 뒤의 괄호에 **(c = getchar()) != EOF** 라고 써져있는것을 볼 수 있다 .

!= 는 **=** 보다 우선 순위가 높기 때문에 **c = getchar()** 를 괄호로 묶어 먼저 진행한다고 보면 된다 .

if문은 뒤 괄호안의 조건을 먼저 확인하고 조건이 참일 경우 if 문의 몸체에 들어간 코드가 실행된다 .

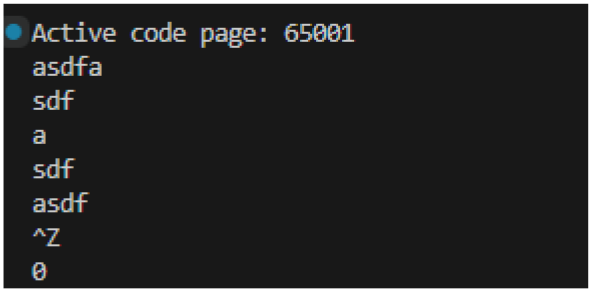
입력 값을 받는 상태일때 Enter를 입력할 경우 우리는 개행이 실행되지만 \n로 입력이 되기 때문에 사용자가 입력한 Enter 횟수를 센다고 보면 된다 .



또한 \n 이 작은 따옴표로 감싸져있는것을 확인할 수 있는데 .

\n이 문자열로써가 아닌 개행으로써 동작하는 값이라는것을 보여주기 위해 작은 따옴표로 감싸준다 .

만약 \n을 큰따옴표로 감쌀 경우



아래와 같이 개행으로써 동작하는 \n을 셀 수 없다 .

> 실습 후 이론

작은따옴표의 역할을 이제서야 알았다 .

1.5.3 실습 문제 3가지

실습 전 이론

예 제 1-8

빈칸, tab과 행의 갯수를 세는 프로그램을 작성하라.

예 제 1-9

한 파일을 읽어서 그 중 빈칸이 연달아 나오면 그것을 모두 한 칸으로 만들어 출력하는 프로그램을 작성하라.

예 제 1-10

한 파일을 읽어서 그 중에 tab이 있으면 그것을 \t, backspace가 있으면 그것을 \b로 그리고 \가 있으면 그것을 \\로 대치해서 출력하는 프로그램을 작성하라.

책에서 실습 문제를 내줬다 .

이전에 만들어 놨던 프로그램은 **EOF** 신호를 받으면 그동안의 문자들을 계산하기 때문에 거기서 조금 변형을 하면 될것 같다 .

실습

1. 첫번째 문제 - 빈칸 , tab, 행의 갯수를 세는 프로그램

```
#include <stdio.h>

int main(){
    int c;
    int blankCount = 0;
    int tabCount = 0;
    int lnCount = 0;
```

```
    while ((c = getchar()) != EOF){
        if(c == '\n')
            ++lnCount;
        if(c == ' ')
            ++blankCount;
        if(c == '\t')
            ++tabCount;
    }
    printf("개행의 갯수는 %d 개 입니다.\n빈칸의 갯수는 %d 개 입니다.\n탭의 갯수는 %d 개 입니다.\n", lnCount, blankCount,
tabCount);
    return 0;
}
```

```
문제   출력   디버그 콘솔   터미널   포트

Active code page: 65001
test    test    test
testtesttest
test test test
tes
tes
test
test
^Z
개행의 갯수는 7 개 입니다.
빈칸의 갯수는 2 개 입니다.
탭의 갯수는 2 개 입니다.
 * 터미널이 작업에서 다시 사용됩니다. 닫으려면 아무 키나 누르세요.
```

기능상 문제는 없는듯

*물론 불필요하게 긴 코드가 존재할 수는 있겠다 .

2. 두번째 문제 - 한 파일을 읽고 그 중 빈칸이 연달아 나오면 그것을 모두 한칸으로 만들어 출력하는 프로그램을 작성

음 if 문으로 공백을 찾고 그 다음 것도 공백일 경우 삭제를 하는 방법으로 해볼까

```
#include <stdio.h>
int main(){
    int c;
    int firstBlank;
```

```
    firstBlank = 0;
    while ((c = getchar()) != EOF){

        if (firstBlank == 0){
            // printf("[!] firstBlank가 0 이므로 putchar를 실행합니다.\n");
            putchar(c);
        }
        if (firstBlank == 1 && c != ' '){
            firstBlank = 0;
            // printf("[!] 공백이 아 니고, firstBlank가 0(거)짓 입니다.\n");
        }
        if (c == ' '){
            firstBlank = 1;
            // printf("[!] 공백이 맞고, firstBlank가 1(참)입 니다.\n");
        }
    }


    return 0;
}
```

공백이 아니면 출력하고 ; 공백이 **getchar**가 입력받을 때까지 반복하도록 ; 그리고 **firstBlank** 변수 공백이 아니면 0 공백이면 1 값을 저장하도록 .

그리고 이제 이 프로그램을 실행하면 두 번 이상 공백이 입력되지 않는 것을 확인할 수 있습니다 .

3. 이스케이 프 문자 - tab은 \t 를 backspace는 \b를 \ 는 \\ 를 출력하는 프로그램

이제 이스케이 프 문자

> 이스 케 이 프

1.5.4 단어 갯수 세기

> 실습 전 이론

이번에는 단어 갯수 세기 프로그램이다 .

하나의 단어를 세는 기준은 빈칸 , **tab,** 혹은 개행이 나오면 한 단어가 끝났다고 생각한다 .

이번 프로그램에선 || 와 상수 지정을 사용하여 행 , 단어 , 문자의 갯수를 센다 .

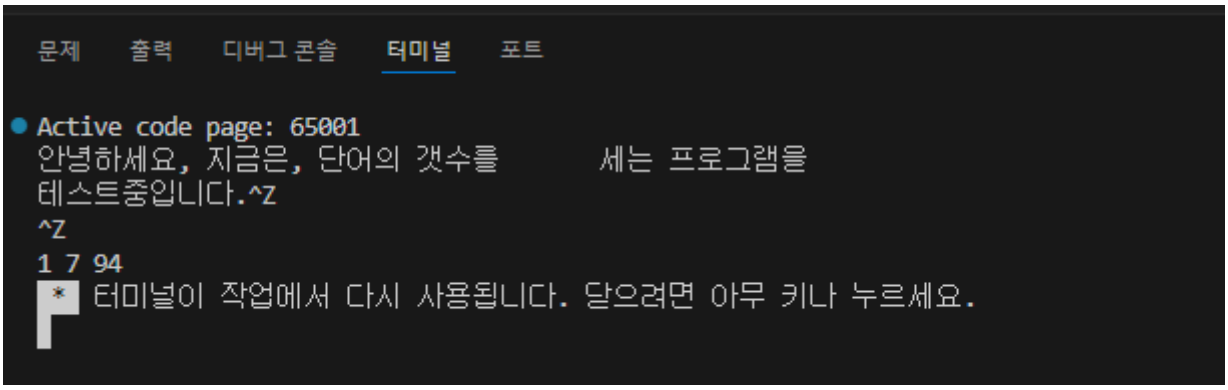
> 실습

```
#include <stdio.h>

#define IN 1
#define OUT 0

int main(){
    int c, nl, nw, nc, state;

    state = OUT;
    nl = nw  = nc = 0;
    while((c = getchar()) != EOF){
        ++nc;
        if (c == '\n')
            ++nl;
        if (c ==' ' || c == '\n' || c =='\t')
            state = OUT;
        else if (state == OUT){
            state = IN;
            ++nw;
        }
    }
    printf("%d %d %d\n", nl, nw, nc);
}
```

변수 **nl, nw, nc** 는 각각 **line, word, count** 라고 보면된다 .

state **out** **(nw)** **1** ,

state **IN** .



EOF , , .

if **||** **or** **if(a || b || c)** **a,b,c** **if**

&& **and**

1.5.5 실습 문제 2가지

> 실습 전 이론

책을 보다보니 이론보다 이거 예제 문제가 본체인것 같다 .

뭔가 프로그램의 원리 혹은 흐름을 이해하지 못하면 턱 하고 막혀버리는 부분들이 존재한다 .

재밌당 .

> 실습 예제

1. 위의 프로그램이 오동작을 하도록 하는 입력파일을 만들어 보라 .

*이 문제는 일단 포기 .. 프로그램 자체가 빡이 나야하는건지 .. 내가 넣은 값이랑 틀린것이 나와야하는건지 .. 추후에 도전 해보기로

2. 파일을 읽어 한 행에 한 단어씩 출력하는 프로그램을 작성해 보라 .

```
#include <stdio.h>

#define IN 1
#define OUT 0

int main(){
    int c, state;

    state = OUT;
    while((c = getchar()) != EOF){
        if (c ==' ' || c =='\t'){
            state = OUT;
            c = '\n';
            putchar(c);
```

```
        }
        else{
        putchar(c);
        }
    }
}
```

```
Active code page: 65001
안녕하세요     지금    은     단어마다       개행으로 출력하도록 만든 프로그램      입니다.
안녕하세요
지금
은
단어마다
개행으로
출력하도록
만든
프로그램
입니다.
```

이 문제는 비교적 좀 쉬웠던것 같다 .

공백 혹은 \t 를 인식하면 해당 문자를 개행문자로 치환해줬다 .

> 실습 후 이론

1.6 배열 (Array)

> 실습 전 이론

파일을 읽어서 그 파일속에 숫자는 몇번이 나오고 빈칸은 몇개가 있는지 , 그 외의 문자들은 몇개인지 세는 프로그램을 작성하려면

하나하나 변수를 지정해야했다 . 변수를 하나씩 지정하는 것보다는 배열을 사용하는게 더 좋다고 한다 .

확실히 어떤 기능 , 문법을 배우려면 그 문법을 배우기 전에 제일 불편하게 일일히 프로그램을 만들어보는게 좋은것 같다 .

예제 풀면서도 하 C는 이런거 어떻게 쓰지 하면서 답답해했었다 .

> 실습

아래 코드는 파일을 읽고 문자를 분류해 각각 몇개인지 확인하는 프로그램이다 .

```
#include <stdio.h>



int main(){
    int c, i, nwhite, nother;
    int ndigit[10];
    nwhite = nother = 0;

    /*배열 안에 요소들을 0으로 지정해주나봄; 이거 일일히 해야되는구나; */
    for (i = 0; i < 10; ++i)
        ndigit[i] = 0;

    while ((c = getchar()) != EOF)
        if (c >= '0' && c <= '9'){
            ++ndigit[c-'0'];
        }
        else if (c == ' ' || c == '\n' || c == '\t'){
```

```
            ++nwhite;
        }
        else {
            ++nother;
        }

        printf("digits = ");

        for(i = 0; i < 10; ++i){
            printf(" %d", ndigit[i]);
        }
        printf(", white space = %d, other = %d\n", nwhite, nother);
}
```

```
int ndigit[10];
```

위 선언이 배열변수 선언이다 .

ndigit[0] 부터 **ndigit[9]** 까지 각각 독립된 변수를 사용할 수 있다 .

여기서 **0~9**를 첨자라고 하고 , 첨자의 시작은 **0**이다 .

```
(c >= '0' && c <= '9')
```

위 코드는 읽어들인 문자가 숫자인지 아닌지를 검사하는 부분이다 .

0부터 **9**안에 포함이 되어있으면 **++ndigit[c-'0'];** 코드가 실행된다 . (? 이부분 이해가 안된다 .)

if (c >='0' && c <='9') ...

라는 부분은 읽어들인 문자가 숫자인지 아닌지를 검사하는 부분이다. 숫자인 경우, 그 숫자가 뭔지를 알려면

c - '0'

을 하면 된다(잘 생각해 보기 바란다). C에서 char와 int형이 섞여 있는 계산을 할 때 모

잘 생각해보라고 한다 .

그리고 책에서의 설명

C에서 char 와 int 형이 섞여있는 계산을할때 모든변수는 int형으로 바뀌게된다는것은 앞에서 설명한바와 같다.

그래서c가'1'이라고 할때 c의 실제값은 1이아니고 문자 1의ascii값이된다.

(물론ascii 외의 다른 문자체계를사용하는컴퓨터의경우는 그 문자체계에 맞는값이 된다).

그래서c에서'0'을빼야 원하는숫자가나오는것이다.

이 숫자(0~9의값이된다)가배열의첨자로사용되고 있다.

생각을 해보고 **ascii** 코드를 찾아봤다 .

48	0x30	0
49	0x31	1
50	0x32	2
51	0x33	3
52	0x34	4
53	0x35	5
54	0x36	6
55	0x37	7
56	0x38	8
57	0x39	9

왼쪽부터 **10**진수 **16**진수 문자 라고 보면되고 , 설명의 중점은

우리가 입력값에 "안녕하세요 저는 **3**을 좋아합니다 . 로 넣었을때 , 우리는 문장속에서 숫자 값을 인식하고 찾아야 한다 .

(c >= '0' && c <= '9') 코드에서 **c** 에 **"3"** 이 들어왔을때 , 문자열을 캐치하고

ndigit[3] 에 우리는 **1**을 더해주어야 한다 **. (++ndigit[c-'0'];** 부분 **)**

그치만 **ndigit[c]** 를 할 경우 **c**에는 문자열이 **3**이 들어가있고 해당 문자열은 **ascii**에서 십진수 **51**로 표현된다 **. (**위의 표 참고 **)**

그러면 우리는 **51**을 **3**으로 바꿔줘야하는데 **51 - 48 = 3 , 48**은 **ascii**에서 문자열 **0**을 뜻한다 .

그리고 **ascii** 문자로써 사용하려면 작은 따옴표를 사용해야하기 때문에

c - '0' = 3이 되고

3이 발견될 때마다

ndigit[3] 의 값에 **1**씩 추가된다 .

후 말이 좀 복잡해졌는데 그래도 너무 재밌네 ;

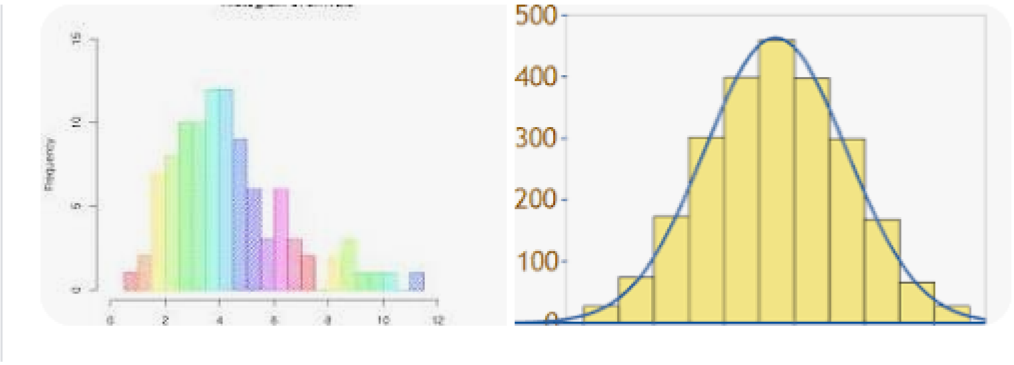
다른 **nwhite** 라던가 **nother** 부분은 이전에 배웠던 부분으로 설명이 되니 넘어간다 .

> 실습 후 이론

1.6.1. 실습 예제 2개

실습 전 이론

히스토그램 - 표로 되어 있는 도수 분포를 정보 그림으로 나타낸 것이다 .



히스토그램에 대한 정의를 잘 몰라서 찾아봤다 . 근데 어떤식으로 진행하면 될지에 대한 정보까지 봐버려서 ; 좀 아쉽네

* 코드는 안보고 어떤식으로 그래프를 표현하는지에 대한 정보를 봤다 .

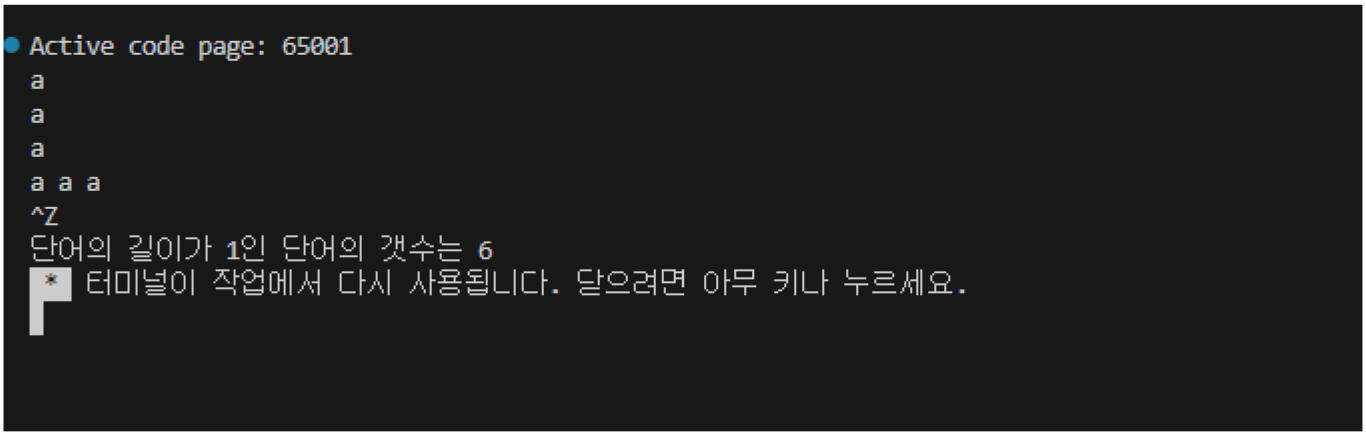
실습 예제

1번 문제 입력된 단어의 길이에 대한 히스토그램을 작성하는 프로그램을 작성하라

```
#include <stdio.h>
#define IN 1
#define OUT 0
#define UPPER 10

int main(){
    int c, i;
    int ndigit[30];
    int wordLength = 0;
    int state;

    /*배열 안에 요소들을 0으로 지정해주나봄; 이거 일일히 해야되는구나; */
```

```
    for (i = 0; i < UPPER; ++i){
        ndigit[i] = 0;
    }

    state = IN;
    while ((c = getchar()) != EOF){
        if (state = IN && c != '\n' && c != '\t' && c != ' '){
            ++wordLength;
        }
        else {
            ++ndigit[wordLength];
            wordLength = 0;
            state = OUT;
        }

    }
    printf("단어의 길이가 1인 단어의 갯수는 %d\n", ndigit[1]);

}
```

한번에 만들기는 너무 복잡해서 일단 테스트용으로 만들었다 . 1개 길이의 단어만 입력하고 출력을 테스트했다 .

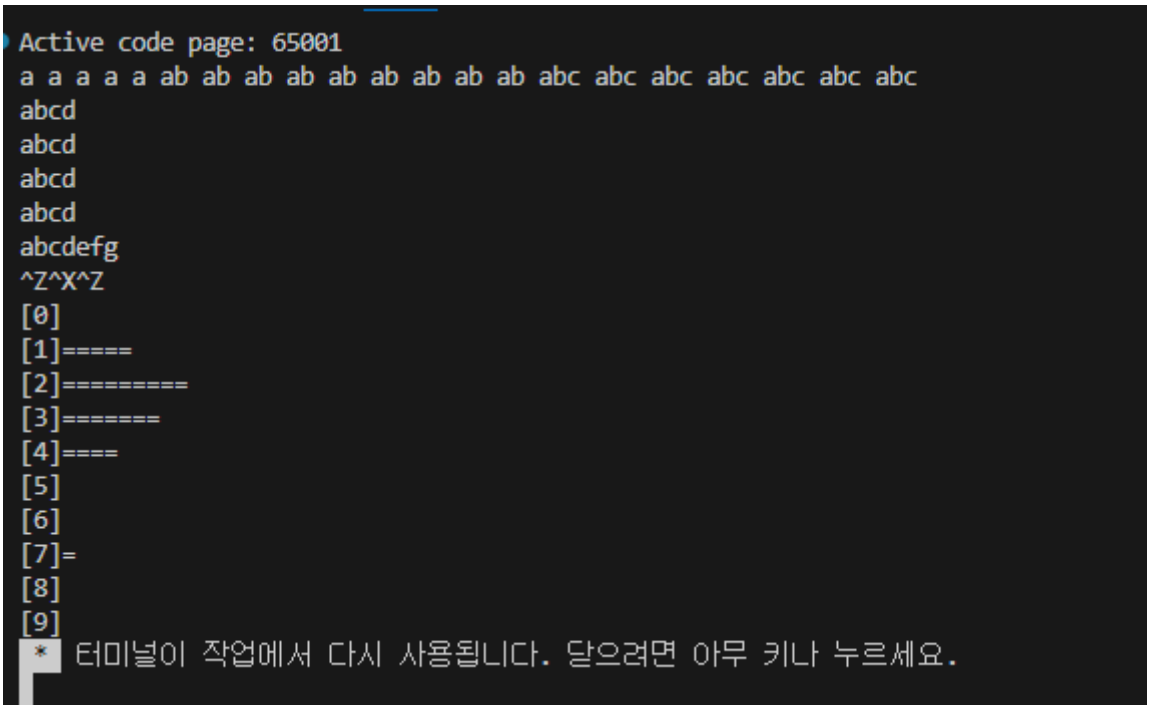


정상적으로 동작하는 것을 확인

```
#include <stdio.h>
#define IN 1
#define OUT 0
#define UPPER 10

int main(){
```

```
    int c, i;
    int ndigit[30];
    int wordLength = 0;
    int text;
    int state;
    int index;

    /*배열 안에 요소들을 0으로 지정해주나봄; 이거 일일히 해야되는구나; */
    for (i = 0; i < UPPER; ++i){
        ndigit[i] = 0;
    }

    state = IN;
    while ((c = getchar()) != EOF){
        if (state = IN && c != '\n' && c != '\t' && c != ' '){
            ++wordLength;
        }
        else {
            ++ndigit[wordLength];
            wordLength = 0;
            state = OUT;
        }

    }

    for (index = 0; index < UPPER; ++index){
        printf("[%d]", index);
        for (text = 0; text < ndigit[index]; ++text){
            printf("=");
        }
        printf("\n");
    }

}
```

위 테스트를 통해 길이 별 단어의 갯수를 파악해 = 문자를 반복하여 출력해서 그래프 모양으로 만들어줬다 .



```
Active code page: 65001
a a a a a ab ab ab ab ab ab ab ab ab abc abc abc abc abc abc abc
abcd
abcd
abcd
abcd
abcdefg
^Z^X^Z
[0]
[1]=====
[2]==========
[3]========
[4]====
[5]
[6]
[7]=
[8]
[9]
* 터미널이 작업에서 다시 사용됩니다. 닫으려면 아무 키나 누르세요.
```

2번 문제 **-** 입력된 문자들의 사용빈도를 나타내는 히스토그램을 출력하는 프로그램을 작성하라

이건 **ndigit[]** 배열의 첨자를 미리 지정하지 않고 새로운 단어가 들어갈때마다 추가해줘야 될까 **?**

그럼 입력 값이 이미 **ndigit** 배열 안에 들어가 있는지를 확인하는 코드 부터 짜봐야겠다 **.**

그러려면 내가 입력한 단어를 배열안에 넣는 코드부터 짜야겠네

근데 **getchar** 는 한 글자씩 가져오는데 **..** 단어를 어떻게 뭉칠까 **..**

배열을 통해 단어를 하나하나 첨자에 넣어야하나 **..** 근데 첨자에 넣으면 그걸 합치는 방법은 아직 안가르쳐줬는데 **..**

***** 아 지금보니까 단어의 사용빈도가 아니라 문자의 사용빈도네 **; a** 몇번썼는지 **b** 몇번썼는지 **..**

이러고 생각보다 쉽다고 생각했는데 이거 좀 생각해보니까 문자를 **ascii**로 치환해서 하는게 나을것같다 **.**

65	0x41	A	97	0x61	a
66	0x42	B	98	0x62	b
67	0x43	C	99	0x63	c
68	0x44	D	100	0x64	d
69	0x45	E	101	0x65	e
70	0x46	F	102	0x66	f
71	0x47	G	103	0x67	g
72	0x48	H	104	0x68	h
73	0x49	I	105	0x69	i
74	0x4A	J	106	0x6A	j
75	0x4B	K	107	0x6B	k
76	0x4C	L	108	0x6C	l
77	0x4D	M	109	0x6D	m
78	0x4E	N	110	0x6E	n
79	0x4F	O	111	0x6F	o
80	0x50	P	112	0x70	p
81	0x51	Q	113	0x71	q
82	0x52	R	114	0x72	r
83	0x53	S	115	0x73	s
84	0x54	T	116	0x74	t
85	0x55	U	117	0x75	u
86	0x56	V	118	0x76	v
87	0x57	W	119	0x77	w
88	0x58	X	120	0x78	x
89	0x59	Y	121	0x79	y
90	0x5A	Z	122	0x7A	z

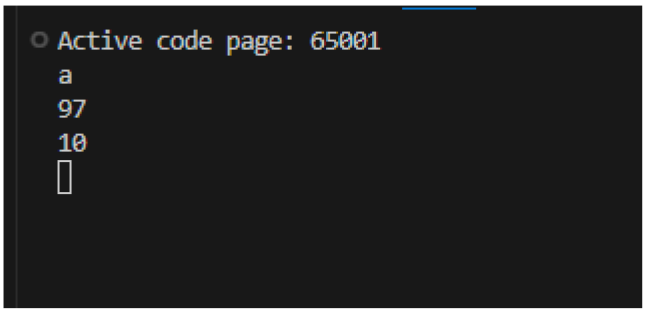
조금 간결하게 하기 위해 소문자만 사용해서 하자

소문자의 갯수는 **26**개 그러면 **ndigit[26]**으로 세팅해두고 시작하자

```
#include <stdio.h>


int main(){
    int c, i;
    int ndigit[26];
    int text;
    int word;

    for (i = 0; i < 26; ++i){
        ndigit[i] = 0;
    }

    while((c = getchar()) != EOF){
        printf("%d\n", c);
```

```
    }


}
```

테스트용 코드 **a** 를 입력하고 **Enter**를 입력해주었다 .



a 는 **ascii**에서 **97** 이고 **ascii**에서 **10**은 개행이다 . **Enter** 키가 **10**으로 출력됐다고 보면 된다 .

ndigit 배열이 **0**부터 **25**까지 존재하니까 **c**에 들어오는 값에서 **97**씩을 빼고 넣어준 뒤 갯수를 하나씩 늘려주면 되겠지

근데 개행까지 들어가니까 입력 받은 값이 소문자 알파벳인지 식별하는 코드도 짜주자

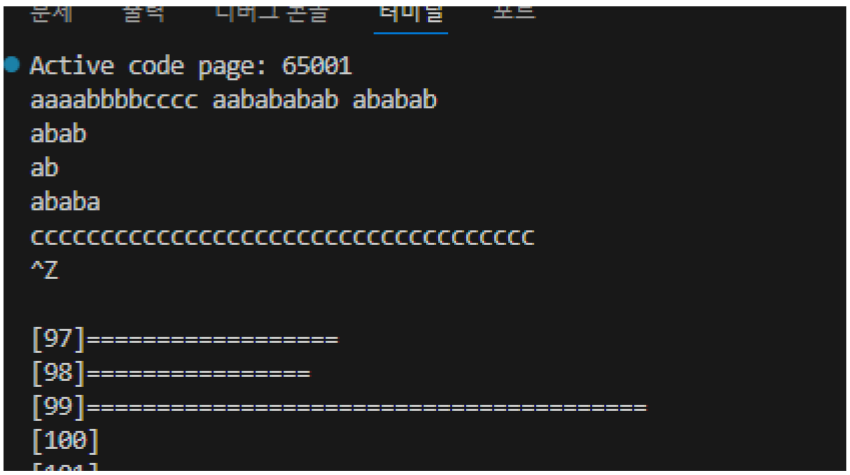
```
while((c = getchar()) != EOF){
        if(c >= 97 && c <= 122){
            c = c-'a';
            printf("%d\n", c);
        }
    }
```

if 문으로 해결 , 이제 들어오는 값의 **ndigit** 인덱스를 **1**씩 더해주면 되겠다 .

```
#include <stdio.h>



int main(){
    int c, i, e;
    int ndigit[26];
    int index;
    int range;
    int alphabet;
    /*배열 안에 요소들을 0으로 지정해주나봄; 이거 일일히 해야되는구나; */

    for (i = 0; i < 26; ++i){
        ndigit[i] = 0;
```

```
    }
    index = 0;
    while((c = getchar()) != EOF){
        if(c >= 97 && c <= 122){
            index = c-'a';
            ++ndigit[index];
            index = 0;
        }
    }


    for (range = 0; range < 26; ++range){
        alphabet = range + 97;
        printf("\n[%d]", alphabet);
        for (e = 0; e < ndigit[range]; e++){
            printf("=");
        }
    }
    printf("%d\n", ndigit[0]);


}
```

어느정도 만들어서



이런 느낌까진 오긴했는데 **... 97**을 다시 어떻게 아스키에서 뜻하는 문자로 바꾸지 **..?**

이후에 방법을 알면 저 부분만 고치면 될것같다 .

> 실습 후 이론

1.7 함수

실습 전 이론

개념자체는 파이썬과 다를게 없다 . 함수 사용 전에 함수에 어떤 값을 넣을지 미리 선언을 한다 정도 ?

기본적으로 함수란 미리 만들어 놓는 기계에 가깝다 .

처음 파이썬을 시작했을땐 왜 함수가 필요한지 잘 몰랐다 . (단순히 10 몇 줄짜리만 만드니 ㅋㅋ)

굳이 함수를 만들어 코드를 넣고 함수를 실행하는 코드를 한번 더 넣는게 이해가 안됐었는데

이제는 안쓰면 너무 복잡하고 찜찜하다 .

기본적으로 함수는 특정 동작을 수행하는 레고 블록에 가깝다 .

로봇으로 비유하자면 팔을 움직이거나 , 옆을보고 , 오른발을 앞으로 뻗는 등

프로그램을 만들다보면 (근본이 워낙 없어서 이게 맞는지는 모르겠다 .)

각각 부위의 부품들을 미리 만든 후 , 최종적으로 조립하는 느낌으로 프로그램을 만들곤 한다 .

1. 엑셀 실행

2. 파일을 불러와 실행

3. 정해진 규칙으로 파일 내용 수정 및 저장

뭐 이런 식으로 3개의 함수를 만들고 최종적으로 세개를 순서대로 실행시키며 프로그램을 완성한다 .

개발자가 아니다보니 업무에 필요한 프로그램을 그때그때 쓸 수 있으면 돼서 , 요정도가 딱 밸런스가 있는것같다 .

실습

```c
#include <stdio.h>

int power(int , int );

int main()
{
    int i;

    for (i = 0; i < 10; ++i)
        printf("%d %d %d\n", i, power(2,i), power(-3,i));
        return 0;
}


int power(int base, int n)
{
    int i, p;

    p = base;
    for (i = 1; i <= n; ++i)
        p = p * base;
    return p;
}
```

main 함수는 프로그램의 가장 중심이 되는 함수 , 파이썬 할때는 저 main함수를 꽤 오랫동안 안썼더랬다 .

밑에 power 함수가 존재하는데 power안에 파라미터로 base와 , n 변수가 들어가며 , "base" 와 "n"이라는 변수명은 함수 안에서만 사용된다 .

함수를 실행시킬때 필요한 정도만 사용하는거로 보면 된다 .

처음엔 왜 그럴까 했는데 , 하다보니 함수마다 변수명을 중복해서 쓰지 못하면 변수명 만들고 기억하는데만 엄청 시간이 들었을것같다 .

power(2,i) 는 그대로 power함수 안의 각각 base, n 이 되어 함수가 실행 후 돌려보낼 결과물을 return 뒤에 담아 보낸다 .

return 이 없는 함수의 경우 void 라는 지칭을 해준다고 들었는데 아직 안나왔으니까 넘어가는걸로

> 실습 후 이론

1.9 문자배열

> 실습 전 이론

배열 중 C에서 가장 많이 사용되는 배열은 문자배열 (아마)

예를 들기 위해 프로그램을 하나 만들어보자

여러 행을 읽고 , 그 중 가장 긴 행을 출력하는 프로그램을 작성해보자

```
while(읽을 행이 있으면)
    if(지금읽는 행이 지금까지 가장 길었던 행보다 길면)
      그 행과 행의 길이를 기억
      가장 긴 행을 출력
```

> 실습

```
#include <stdio.h>
#define MAXLINE 1000

int getline(char line[], int maxline);
void copy(char to[], char from[]);

int main()
{
    int len;
    int max;
    char line[MAXLINE];
    char longest[MAXLINE];
    max = 0;
    while ((len= getline(line,MAXLINE)) > 0 )
        if (len > max) {
            max = len;
            copy(longest, line);
        }
        if (max > 0)
```

```
            printf("%s", longest);
        return 0;
}

int getline(char s[], int lim)
{
    int c, i;
    for(i=0; i<lim-1 && (c=getchar()) !=EOF && c != '\n'; ++i)
        s[i] = c;
    if (c == '\n') {
        s[i] = c;
        ++i;
    }
    s[i] = '\0';
    return i;
}

void copy(char to [], char from[]){
    int i;

    i = 0;
    while ((to[i] = from[i]) != '\0')
        ++i;
}
```

실습에 사용되는 예제 코드는 위와 같다 .

(이대로 실행하면 오류가 난다 . **getline** 이라는 함수가 이미 라이브러리에 있기 때문이니까 임의로 변경하여 작성해주자)

코드가 좀 길다보니 쓰는데 시간이 좀 걸리긴 했는데 .. 흠 어떤 방식으로 이해를 할까 하다가 , 너무 디테일하게 정확한 내용을 처음부터 알기는 어려우니 대략적으로 함수별로 파악해보자

MAXLINE = 1000으로 고정

getline 함수 : 문자배열 **s,** 정수 **lim** 파라미터를 받아서 **MAXLINE -1** 보다 작고 **EOF** 이 아니면서 **\n** 이 아닐경우엔 **s**의 **i**번째 배열에 **c**를 넣는다 .

만약 **\n**일 경우 **s[i]** 에 **\n** 을 넣고 **i** 에 **1**을 더한다 **? (**근데 **for**문에서 이미 **c != '\n'** 조건에 거짓이 되는데 어떻게 **if** 문까지 닿지 **?)**

아 **for** 문은 그냥 바로 다음 줄인 **s[i] = c;** 까지 였네 **;**

간단하게 설명하면 입력 값을 받고 그걸 문자배열 **s** 에 저장 **,** 배열의 마지막에는 **null** 선언 **,** 그리고 몇 글자인지 **(i)** 를 리턴

main 함수 **: getline**을 실행해서 리턴한 값이 지금까지 받은 값중 가장 긴 문자인지 확인 **,** 가장 길다면 지금 입력 값은 최댓값 **(max)**로 선언한 뒤 **copy** 함수 실행

copy 함수 **:** 입력 값을 받아서 **\0**이 나올때 까지 **to** 문자배열에 하나씩 선언해준다 **. (**복사하는 방식이 이렇게 되는구나 **..)**

파이썬에서는 그냥 **i = t** 이런식으로 바로 문자열이 복사가 됐었는데 **C**에서는 인덱스 하나하나에 한글자씩 복사를 하는게 놀랍다 **..**

> 실습 후 이론

위 코드에서 for문이 어디까지인지 , while문이 어디까지 포함인지를 잘 인지를 못해서 좀 헤맸다 .

파이썬은 들여쓰기로 인식을 했었는데 C는 그게 아니라 ... 코드를 좀 더 유심히 봐야겠다 .

1.10 외부변수

> 실습 전 이론

이전 챕터에서 만든 프로그램의 경우 **main**에 **longest** 등의 변수가 있었다 .

이 변수들은 **main** 함수 내에서만 의미를 가진다 .

같은 이름의 변수를 다른 함수에서 쓰더라도 , 전혀 별개의 값이 된다 . 이를 지역 **(local)** 변수라고 한다 .

여러 함수에서 공통으로 사용하는 함수를 공통 변수 **(global)**라고 한다 .

공통 변수를 정의할때는 **extern** 이라는 명령어를 사용한다 .

해당 변수는 함수의 바깥에서도 정의하고 , 그 변수를 사용하는 각 함수에서도 정의되어야 한다 .

이전에 만든 가장 긴 행을 찾는 프로그램의 변수에서 **longest**와 **max** 그리고 **line**의 세 변수를 공통으로 사용하도록 프로그램을 고치면 다음과 같다 .

> 실습

```
#include <stdio.h>

#define MAXLINE 1000

int max;
char line[MAXLINE];
char longest[MAXLINE];

int my_getline(void);
void copy(void);

int main()
{
    int len;
    extern int max;
    extern char longest[];
```

```
    max = 0;

    while ( ( len = my_getline()) > 0)
        if (len > max) {
            max = len;
            copy();
        }

    if(max > 0)
        printf("%s", longest);
    return 0;
}

int my_getline(void)
{
    int c, i;
    extern char line[];

    for(i = 0; i < MAXLINE-1 && (c=getchar()) != EOF && c != '\n'; ++i)
        line[i] = c;
    if (c == '\n'){
        line[i] = c;
        ++i;
    }
    line[i] ='\0';
    return i;
}

void copy (void)
{
    int i;
    extern char line[], longest[];
    i = 0;
    while ( ( longest[i] = line[i]) != '\0')
        ++i;
}
```

기능은 이전 게시글과 동일 하다 . 다만 차이점으로는 실행하는 함수에 같이 전달하는 파라미터가 없는 것

extern 명령어를 통해 외부에 있는 변수를 같이 사용하고 있다 .

함수에 같이 전달하는 파라미터가 **void**로 변경이 된 것을 확인할 수 있다 .

함수를 **void**로 선언한 **copy**의 경우 리턴 값이 없다는 뜻 ,

파라미터에 들어간 **void**는 " 이 함수는 파라미터를 절대 받지 않는다 " 라는 뜻

> 실습 후 이론

프로그램을 작성할때 그럼 모든 변수를 공통으로 선언하면 편하지 않나 라는 생각이 들지도 모르지만 ,

프로그램 규모가 커지고 비슷한 변수들이 늘어나면 분리하는게 더 유용하지 않나 싶다 . 장단점이 있는듯

1장은 여기서 마무리 , 함수 관련하여 문제가 몇가지 있는데 이것들은 추후에 풀이를 같이 올려봐야겠다 .