시스템 프로그래밍 기초

Introduction to System Programming

ICT융합학부 조용우

0. Starting from Zero

Starting from 0

- 0
  - C의 자연스러운 시작점
  - 숫자를 0부터 셈
  - 0은 거짓, 0아닌 수는 참
  - 배열의 첫 원소
  - 문자열의 끝
  - 포인터의 NULL 값
  - 외부/정적 변수의 초기화 값

C?

- C언어
  - → 원하는 결과를 얻기 위한 프로그램 작성시 필요한 일종의 언어
  - → UNIX 운영체제 하에서 시스템 프로그래밍을 하기위해 개발됨
  - → 작은 언어, 구조적인 언어, 간결성, 강력한 기능, 빠른 속도, 이식성, 모듈성
- C로 개발된 운영체제들
  - → UNIX, OS/2, MS-DOS, Windows, LINUX, OSX, etc.

C?

- C 이전의 역사
  - → ALGOL60 (ALGOrithmic Language) 1960년 국제위원회에서 발표
    - 구조적 프로그래밍 언어
  - → CPL (Combined Programming Language): 1963년 영국 케임브리지 대학
  - → BCPL (Basic CPL): 1967년 영국 케임브리지 대학의 Martin Richards
    - 형이 없는 시스템 프로그래밍 언어, 다른 언어의 컴파일러 작성용
    - 기본 자료형은 기계 워드, 포인터와 주소연산이 많이 사용됨
    - "Before C Programming Language"
  - → UNIX: 1969년 미국 AT&T Bell 연구소의 Dennis Ritch와 Ken Tompson이 PDP-7용 운영체제로 어셈블리어를 사용하여 개발
  - → B: 1970년 미국 AT&T Bell 연구소의 Ken Tompson이 PDP-11 UNIX재작성용으로 개발

C?

- C의 역사
  - → "C"
    - 1972년 미국 AT&T Bell 연구소의 Dennis Ritche가 PDP-11 UNIX기반에서 개발
    - 1973년 UNIX kernel을 전부 C로 재작성 (어셈블리를 벗어난 최초의 운영체제)
  - → "The C Programming Language": 1978년 Kernighan과 Ritche가 출판
  - → ANSI C: 1990년 표준화 완료된 "the standard C"
  - → C++: 표준 C에 OOP (Object Oriented Programming, 객체지향프로그래밍) 개념을 추가

1. An Overview of C

프로그래밍과 준비

- 프로그램 작성
  - → $vim *sourcename*.c
- 컴파일
  - → $gcc *sourcename*.c
  - → 결과 (목적 파일): a.out
- 목적 파일명 지정
  - → $gcc -o *execname sourcename*.c
- 실행
  - → $./a.out

프로그램 출력

```
/* sea.c */
#include <stdio.h>
int main(void)
{
    printf("from sea to shining C\n");
    return 0;
}
```

```
from sea to shining C
```

프로그램 출력

```
$gcc sea.c
```

- 앞에서 작성한 프로그램을 컴파일
- 코드에 오류가 없다면 실행 파일 (목적 파일)인 a.out이 생성

```
$./a.out
from sea to shining C
```

- 프로그램 수행 결과

Dissection of the sea Program

```
/* sea.c */
```

- /* */은 주석을 나타내며, 컴파일러는 이것을 무시함

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

- #으로 시작하는 행은 전처리기(preprocessor)에서 처리됨
- 프로그램을 컴파일하면 컴파일 전에 전처리기가 먼저 동작됨
- printf() 함수 사용 위해 <stdio.h> 헤더 파일 복사
- 필요하다면 헤더파일을 지정한다

Dissection of the sea Program

```
int main(void)
```

- 식별자 다음에 괄호 ()가 오면 그 식별자는 함수라는 것을 나타냄
- 프로그램은 함수로 구성됨
- 프로그램의 수행은 항상 main() 함수로부터 시작함
- main() 함수 정의
  - → void - 인수를 받지않음
  - → int형 return값을 가짐

Dissection of the sea Program

{ }

- 중괄호는 여러 문장들을 그룹화하기 위해 사용됨
- 즉, 중괄호로 둘러싸인 것은 하나의 단위로 취급됨

printf()

- 화면 출력기능을 가진 library 함수, stdio.h

"from sea to shining C\n"

- 큰 따옴표로 둘러싸인 일련의 문자들을 문자열 상수라고 함
- 문자열 상수를 이루는 단어들은 그 본래의 의미를 잃어버림
- \n은 개행(new line) 문자를 나타냄 → 프로그래머가 직접 지시해야함
- 프로그램 상에서 일반 문자로 표현할 수 없는 것을 표현하고 싶을 때 역슬래시 \와 결합된 문자를 사용함

Dissection of the sea Program

```
printf("from sea to shining C\n");
```

- C에서 모든 문장은 세미콜론으로 끝남

```
return 0;
```

- 0값을 운영체제에 return

printf()

- 화면에 출력하는 함수
- 연속적으로 printf()가 있을 경우, 뒤에 나오는 printf()의 출력은 바로 앞 printf()의 마지막 출력 위치에서부터 시작하여 출력한다.
- 즉, 예제의 printf("from sea to shining C\n");는 다음 코드와 동일한 내용을 출력한다.

```
printf("from sea to ");
printf("shining C");
printf("\n");
```

printf() example

```
/* sea2.c */
#include <stdio.h>
int main(void)
{
    printf("\n\n\n\n\n\n\n\n\n\n");
    printf("            **************************\n");
    printf("            *  from sea              *\n");
    printf("            *  to shining C          *\n");
    printf("            **************************\n");
    printf("\n\n\n\n\n\n\n\n\n\n");
    return 0;
}
```

```
            **************************
            *  from sea              *
            *  to shining C          *
            **************************
```

C에는 행의 개념이 없음

```
/* sea.c */
#include <stdio.h>
int main(void)
{printf("from sea to shining C\n"); return 0;}
```

```
from sea to shining C
```

변수, 수식, 배정

```
/* The distance of a marathon in kilometers. */

#include <stdio.h>

int main(void)
{
    int     miles, yards;
    float   kilometers;

    miles = 26;
    yards = 385;
    kilometers = 1.609 * (miles + yards / 1760.0);
    printf("\nA marathon is %f kilometers.\n\n", kilometers);
    return 0;
}

A marathon is 42.185970 kilometers.
```

Dissection of the marathon Program

```
int miles, yards;
```

- 선언문: 변수 miles, yards는 정수값을 가지는 변수

```
float kilometers;
```

- 선언문: 변수 kilometers는 실수값(유효숫자 6자리)을 가지는 변수
- 모든 변수는 선언하고 나서 사용

```
miles = 26;
yards = 385;
```

- 배정문: 정수형 상수 26과 385가 변수 miles와 yards에 배정

Dissection of the marathon Program

```
kilometers = 1.609 * (miles + yards / 1760.0);
```

- 배정문
- *, +, /: 연산자 (-, %, …)

```
printf("\nA marathon is %f kilometers.\n\n", kilometers);
```

- 변환형식 %f와 인자 kilometers는 짝을 이루며, kilometers의 값이 보통 소수점(float) 형식 %f의 위치에 출력됨
- 변수의 값을 출력하려면 서식지정이 필요함
- 수식의 변환 규칙 (conversion rule)
  - 7/2 → 3
  - 7.0/2 → 3.5

#define과 #include의 사용

```
#define LIMIT 100
#define PI 3.14159
#define C 299792.458 /* speed of light in km/sec */
```

- #: 전처리기 지시자 (preprocessing directive)
- LIMIT, PI, C: 심볼릭 상수(symbolic constant)

```
#include "my_file.h"
```

- 코드에 my_file.h 파일의 사본 포함
- C에서 제공하는 표준 헤더파일
  - → stdio.h, string.h, math.h, …, <xxx.h>

#define과 #include의 사용

```
/* pacific_sea.h */
#include <stdio.h>

#define   AREA                        2337
#define   SQ_MILES_PER_SQ_KILOMETER   0.38610215854244458
#define   SQ_FEET_PER_SQ_MILE         (5280 * 5280)
#define   SQ_INCHES_PER_SQ_FOOT       144
#define   ACRES_PER_SQ_MILE           640
```

#define과 #include의 사용

```
/* pacific_sea.c */
#include "pacific_sea.h"

int main(void)
{
    const int    pacific_sea = AREA;     /* in sq kilometers */
    double       acres, sq_miles, sq_feet, sq_inches;

    printf("\nThe Pacific Sea covers an area");
    printf(" of %d square kilometers.\n", pacific_sea);
    sq_miles = SQ_MILES_PER_SQ_KILOMETER * pacific_sea;
    sq_feet = SQ_FEET_PER_SQ_MILE * sq_miles;
    sq_inches = SQ_INCHES_PER_SQ_FOOT * sq_feet;
    acres = ACRES_PER_SQ_MILE * sq_miles;
    printf("In other units of measure this is:\n\n");
    printf("%22.7e acres\n", acres);
    printf("%22.7e square miles\n", sq_miles);
    printf("%22.7e square feet\n", sq_feet);
    printf("%22.7e square inches\n", sq_inches);
    return 0;
}
```

Dissection of the pacific_sea Program

```
const int pacific_sea = AREA; /* in sq kilometers */
```

- const: ANSI C에 소개된 한정자, 초기화 이후 값 변경 불가

```
double acres, sq_miles, sq_feet, sq_inches;
```

- double: 유효숫자 15자리 (float는 6자리)

```
printf("%22.7e acres\n",acres);
5.7748528e+05 acres
```

- 5.7748528×10^5
- 다음 장 참조

printf()와 scanf()의 사용

printf("서식지정문자열", 변수);

- 화면출력용 함수
- printf()는 출력된 문자의 수를 int형으로 리턴
  (오류 발생 시 음수값 리턴)
- 서식지정 문자열
  → 일반문자열, 변환문자열(%), 확장문자열(\)

printf("%변환문자", 변수);

- printf()의 변환문자열

printf()와 scanf()의 사용

printf() 변환문자	
c	as a character (문자)
d	as a decimal integer (10진 정수)
ld	as a long type decimal integer (long형 10진 정수)
e	as a floating point number in scientific notation (지수형)
f	as a floating point number (float, double)
g	in the e-format or f-format, whichever is shorter
s	as a string (문자열)

printf()와 scanf()의 사용

```
printf("abc");
printf("%s", "abc");
printf("%c%c%c", 'a','b','c');
```

- 화면에 abc 출력
- 'a'는 소문자 a에 해당하는 문자 상수이다.

printf()와 scanf()의 사용

변환문자의 옵션 지정		
%[필드폭].[자릿수][변환문자]		
%d	→	123
%5d	→	␣␣123
%10d	→	␣␣␣␣␣␣␣123
%2d	→	123 (지정 필드폭의 칸수가 필요한 자릿수보다 작아도 필요한 숫자는 모두 출력)
%f	→	654.321000 (표준출력, 소수점 이하 6자리)
%12f	→	␣␣654.321000 (12칸에 출력, 소수점 이하는 6자리로 표준출력)
%9.2f	→	␣␣␣654.32 (9칸에 출력, 소수점 이하는 2자리로 출력)

printf()와 scanf()의 사용

scanf("서식지정문자열", 변수주소);

- 키보드 입력용 함수
- scanf()함수는 성공적으로 입력된 횟수를 int형을 리턴

scanf("%d", &x);

- &기호는 주소연산자로 &x는 "x의 주소"라고 읽음
- %d는 x가 해석될 방식에 상응하는 형식으로, 입력 문자열을 10진 정수로 해석하여 x의 주소에 결과값을 저장함

The Use of printf() and scanf()

scanf() 변환문자	
c	to a character (문자)
d	to a decimal integer (10진 정수)
ld	to a long type decimal integer (long형 10진 정수)
f	to a floating point number (float)
lf	to a floating point number (double)
LF	to a floating point number (long double)
s	to a string (문자열)

printf()와 scanf()의 사용

```
/* echo.c */
#include <stdio.h>

int main(void)
{
    char    c1, c2, c3;
    int     i;
    float   x;
    double  y;

    printf("\n%s\n%s", "Input three characters,"
        "an int, a float, and a double: ");
    scanf("%c%c%c%d%f%lf", &c1, &c2, &c3, &i, &x, &y);
    printf("\nHere is the data that you typed in:\n");
    printf("%3c%3c%3c%5d%17e%17e\n\n", c1, c2, c3, i, x, y);
    return 0;
}
```

printf()와 scanf()의 사용

```
Input three characters,
an int, a float, and a double:  ABC␣␣3␣␣55␣␣77.7
Here is the data that you typed in:
  A  B  C    3     5.500000e+01     7.770000e+01
```

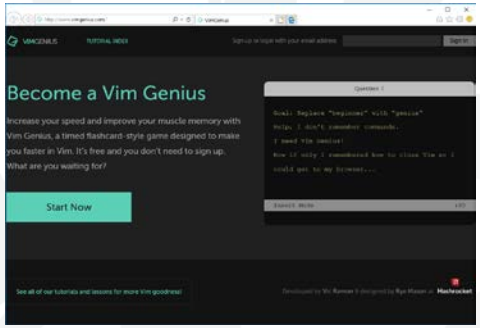
printf()와 scanf()의 사용

```
scanf("%c%c%c%d%f%lf", &c1, &c2, &c3, &i, &x, &y);
```

- 숫자를 입력 받을 때, 여백(공백, 탭, 개행)은 건너뜀
- 하지만, 문자를 입력 받을 때는 건너뛰지 않음
- AB␣C␣3␣55␣77.7로 입력하면 제대로 실행되지 않음
- 세번째 문자로 ' '(공백문자)를 읽고나서 'C'를 10진 정수로 읽고자 하기 때문에 문제가 됨

Vim Games

- VIM Adventure
  - → https://vim-adventures.com/
- VimGenius
  - → http://www.vimgenius.com/
- Interactive Vim tutorial
  - → http://www.openvim.com/



Flow of Control

- 선택
  - → if
  - → if-else
- 반복
  - → for
  - → while
  - → do-while

if 문

- 일반적인 형태

```
if (expr)
    statement
```

  - → 조건식(*expr*)이 참(true)이면 문장(*statement*) 실행
  - → false: zero, true: non-zero
  - → 단문이면 {} 생략

```
a = 1;
if (b == 3)
    a = 5;
printf("%d", a);
```

  - → ==는 *is equal to*
  - → b가 3이면 a=5
  - → b가 3이 아니면 문장 (a=5) 실행 안함, printf() 문 실행 시 1 출력

if-else 문

- 일반적인 형태

```
if (expr)
    statement1
else
    statement2
```

  - → 조건식이 참이면 (*expr*이 0이 아니면) 문장1 실행, 그렇지 않으면 문장2 실행
  - → 여러 문장을 포함해도 if-else문 전체가 하나의 문장

예문

```
if (cnt == 0) {
    a = 2;
    b = 3;
    c = 5;
}
else {
    a = -1;
    b = -2;
    c = -3;
}

printf("%d", a + b + c);
```

→ cnt 가 0값을 가지면 10 출력, 그렇지 않으면 -6 출력

while 루프

- 일반적인 형태

```
while (expr)
    statement
```

→ 조건식이 참이면 (*expr*이 0이 아니면) 문장 실행 후, while 루프 처음으로 복귀, *expr*이 0이 될 때까지 반복

while 루프

```
/* consecutive_sums.c */
#include <stdio.h>

int main(void)
{
    int  i = 1, sum = 0;
    while (i <= 5) {
        sum += i;
        ++i;
    }
    printf("sum = %d\n", sum);
    return 0;
}
```

Dissection of the consecutive_sum Program

```
while (i <= 5) {
    sum += i;
    ++i;
}
```

→ <=는 less than or equal to

```
variable op= expr
variable = variable op expr
```

- sum += i;
  - → sum = sum + i;
- ++i;
  - → ++i 증가 --i; 감소
  - → i = i + 1; i = i - 1;
  - → p85. 2.10 Increment and Decrement Operators 참조

for 루프

- 일반적인 형태

```
for (expr1; expr2; expr3)
    statement
```

```
expr1;
while (expr2) {
    statement
    expr3;
}
```

  - expr1 초기화 배정한 후, expr2를 검사하여 0이 아닌 경우, statement를 수행한 후, expr3으로 저장 값을 증가시키고 다시 expr2 검사하면서 0이 아닌 동안 반복
  - expr3이 루프에서 가장 마지막에 실행됨

for 루프

```
#include <stdio.h>
int main(void)
{
    int sum=0;
    for (i=1; i<=5; ++i ) {
        sum+=i;
    }
    printf("sum= %d \n", sum);
    return 0;
}
```

함수

- C프로그램은 여러 파일들을 가질 수 있으며, 각각의 파일은 여러 함수들을 가질 수 있음
- main()함수
  → 이 함수로부터 프로그램 시작, main()함수에서 다른 함수가 호출되어 프로그램이 구성됨

```
int main(void)
{
     ...
     subfunction1(arguments);
     ...
}

...

subfunction1(arguments)
{
     ...
}
```

함수

- 함수 원형 (function prototype)

```
type function_name(parameter type list);
```

  - 함수는 사용되기 전 선언되어야 하는 데, 이런 함수선언형식을 함수원형이라함
  - 컴파일러는 함수원형을 통해 함수에 전달될 인자의 수와 형, 그리고 함수에서 리턴될 값의 형을 알 수 있음
  - function_name이 함수의 이름, type형의 리턴값을 가짐, parameter type list는 콤마로 분리된 형들의 목록
  - 이 목록에서 식별자 사용은 옵션 (함수 원형에 영향없음)
  - 인자 혹은 리턴값이 없을 경우 void 사용
  - 인자의 개수가 가변적일 때에는 ... 사용

- 예제 (stdio.h에 정의된 printf()의 원형)

```
int printf(const char * format, ...);
```

1.7 Functions

```c
/* maxmin.c */
#include <stdio.h>

float    maximum(float x, float y);
float    minimum(float x, float y);
void     prn_info(void);

int main(void)
{
    int      i, n;
    float    max, min, x;

    prn_info();
    printf("Input n:    ");
    scanf("%d", &n);
    printf("\nInput %d real numbers:  ", n);
    scanf("%f", &x);
    max = min = x;
    for (i = 2; i <= n; ++i) {
        scanf("%f", &x);
        max = maximum(max, x);
        min = minimum(min, x);
    }
    printf("\n%s%11.3f\n%s%11.3f\n\n",
        "Maximum value:", max,
        "Minimum value:", min);
    return 0;
}
```

1.7 Functions

```
float maximum(float x, float y)
{
    if (x > y)
        return x;
    else
        return y;
}

float minimum(float x, float y)
{
    if (x < y)
        return x;
    else
        return y;
}

void prn_info(void)
{
    printf("\n%s\n%s\n\n",
        "This program reads an integer value for n, and then",
        "processes n real numbers to find max and min values.");
}
```

함수의 선언과 정의

- 함수 선언 (function declaration)

```
float   maximum(float x, float y);
```

  - → 컴파일러에게 maximum()함구가 2개의 float형 인자를 가지고, 리턴값은 float형이라는 것을 알려줌

- 함수 정의 (function definition)

```
float maximum(float x, float y)
{
    if (x > y)
        return x;
    else
        return y;
}
```

  - → 이 함수가 호출될 때, 실제 실행될 작업을 명확하게 기술

Call-by-value

```
#include <stdio.h>

int main(void)
{
    int   a = 1;
    void  try_to_change_it(int);

    printf("%d\n", a);     /* 1 is printed */
    try_to_change_it(a);
    printf("%d\n", a);     /* 1 is printed again! */
    return 0;
}

void try_to_change_it(int a)
{
    a = 777;
}
```

배열 (Arrays)

- C에서 문자열(string)은 문자(character)의 배열(array)이고 배열이름 자체가 하나의 포인터(pointer)임
- 배열(Arrays)
  - → 배열은 동일한 형을 갖고 개수가 많은 변수가 요구될 때 사용

```
int a[3]
```

- 이 배열은 int형 원소 a[0], a[1], a[2]로 구성
- 배열의 첨자는 항상 0부터 시작

배열

```
/* sorting program */
...
int i, j, score[CLASS_SIZE], sum=0, tmp;
printf("Input %d scores: ", CLASS_SIZE);
for(i = 0; i < CLASS_SIZE; ++i) {
    scanf("%d", &score[i]);
    sum += score[i];
}
for(i = 0; i < CLASS_SIZE - 1; ++i) {
    for(j = CLASS_SIZE - 1; j > i; --j) {
        if(score[j-1] < score[j]) {
            tmp = score[j-1];
            score[j-1] = score[j];
            score[j] = tmp;
        }
    }
}
```

배열

```
Input 5 scores: 63 88 97 53 77
ordered scores :
    score[0] = 97
    score[1] = 88
    score[2] = 77
    score[3] = 63
    score[4] = 53
378 is the sum of all the scores
75.6 is the class average
```

배열

- 버블 정렬(bubble sort)은 정수들을 정렬하기 위해 사용됨
- 현재 비교된 원소들의 순서가 맞지 않으면, 그들의 값은 상호 교환됨
- 여기에서 이 상호 교환은 다음 코드에 의해 이루어짐

```
tmp = score[j-1];
score[j-1] = score[j];
score[j] = tmp;
```

문자열

- C에서 문자열은 문자의 배열
- 이 절에서는 문자열의 사용법과 아울러 getchar() 과 putchar() 의 사용법을 소개
- 다음 프로그램은 문자 배열(문자열)에 사용자가 입력한 라인을 저장한 후, 화면에 입력의 역순으로 출력하는 프로그램

문자열

```
...
printf("\nHi! what is your name?   ");
for(i = 0; (c = getchar()) != '\n'; ++i) {
    name[i] = c;
    if (isalpha(c))
        sum += c;
}
name[i] = '\0';
printf("\n %s%s%s \n%s",
    "Nice to meet you  ", name, ".",
    "Your name spelled backwards is  ");
for(--i; i >= 0; --i)
    putchar(name[i]);
    printf("\n%s%d%s\n\n%s\n",
        "and the letters in your in your name sum to", sum, ".",
        "Have a nice day !");
...
```

문자열

```
Hi! what is your name?  Alice B. Carole
Nice to meet you Alice B. Carole.
Your name spelled backwards is eloraC .B ecilA
and the letters in your name sum to 1142.
Have a nice day!
```

포인터 (Pointers)

- 포인터는 메모리에 있는 한 대상의 주소
- 배열명은 그것 자체가 하나의 포인터
- 다음에 나오는 프로그램은 배열과 포인터의 관계에 대해 설명하기 위해 설계됨

포인터

```
/* the relationship between arrays and pointers */
...
char    c='a', *p, s[MAXSTRING];

p = &c;
printf("%c%c%c  ", *p, *p + 1, *p + 2);
strcpy(s, "ABC");
printf("%s  %c%c%s\n", s, *s + 6, *s + 7, s + 1);
strcpy(s, "she sells sea shells by the seashore");
p = s + 14;
for( ; *p != '\0'; ++p) {
    if(*p == 'e')
        *p='E';
    if(*p == ' ')
        *p='\n';
}
printf("%s\n", s);
...
```

Result

```
abc  ABC  GHBC
she sells sea shElls
by
thE
sEashorE
```

배열

- C에서 배열, 문자열, 포인터는 밀접하게 관련되어 있음

```
char *p, s[100];
```

- 'p' 가 포인터 변수인데 반해, s는 s[0]을 포인트하는 포인터 상수
- s[i] 와 *(s + i)는 동등
- 이와 유사하게 p[i] 와 *(p + i)도 동등

파일

```
#include <stdio.h>

int main(void)
{
    int  c;
    FILE *ifp;

    ifp = fopen("my_file", "r");
    ...
```

파일

- Main()의 몸체에서 두 번째 줄은 FILE형 포인터인 ifp(“infile pointer”의 약자)를 선언한다. FILE 형은 특별한 구조로stdio.h 에 정의되어 있음

```
ifp = fopen("my_file", "r");
```

- fopen() 함수는 인자로 두 개의 문자열을 취하며, FILE형 포인터를 리턴함
- 첫 번째 인자는 파일명이고, 두 번째 인자는 파일이 오픈될 모드
- fopen() 함수는 표준 라이브러리에 존재하며, 이것이 함수원형은 stdio.h 에 있음
- 만약 어떤 이유로 파일에 접근할 수 없다면, fopen()은 NULL 포인터를 리턴

명령어라인인자

- 명령어 라인 인자를 프로그램 안에서 받아들일 수 있는 방법

```
#include <stdio.h>
int main(int argc, char *argv[])
{
    ...
```

- 인자 argc는 "인자숫자(argument count)"를 의미
- 인자 argv는 "인자변수(argument variable)"를 의미

대문자 개수 세는 프로그램

```
/* count uppercase letters in a file. */
...
int main(int argc, char *argv)
{
    int  c, i, letter[26];
    FILE *ifp, *ofp;
    if(argc != 3) { ... }
    ifp = fopen(argv[1], "r");
    ofp = fopen(argv[2], "w");
    for (i = 0; i < 26; ++i)
        letter[i] = 0;
    while ((c = getc(ifp)) != EOF)
        if (c>='A' && c<='z')
            ++letter[c-'A'];
    for (i = 0; i < 26; ++i) {
        if (i % 6 == 0)
            putc('\n', ofp);
        fprintf(ofp, "%c:%5d    ", 'A' + i, letter[i]);
  }
  putc('\n', ofp);
  ...
```

대문자 개수 세는 프로그램

- 다음과 같은 명령어를 입력시켜 보자.
- cnt_letters *chapter1 data1*
- data1 파일에서 다음과 같은 결과를 얻을 수 있을 것이다.

```
A:  248 B:  240 C:  1292 D:   100 E:  461 F:   79
G:   33 H:   51 I:   569 J:     1 K:    3 L:   20
M:   71 N:  108 O:   164 P:  1222 Q:   19 R:   18
S:  215 T:  455 U:    33 V:    27 W:   63 X:   87
Y:   14 Z:   17
```

운영체제

- C 프로그램의 작성과 실행

1. 편집기를 이용하여 C프로그램이 작성될 pgm.c 라는 문서파일을 만든다. 파일명은 반드시 .c로 끝나야 한다.

```
$vim pgm.c
```

2. 프로그램을 컴파일한다.

```
$gcc pgm.c
```

3. 프로그램을 실행한다.

```
$./a.out
```

UNIX

```
$gcc -o sea sea.c
```

- UNIX에서 확장자 .c를 삭제하여 소스파일과 실행 파일의 이름을 동일하게 하는 것이 일반적임
- 문법오류들은 컴파일러에 의해 검사되고, 실행시간 오류들은 프로그램이 실행되는 동안에 발생됨
- 예를 들어, 만약 0으로 나누기를 시도한 문장들이 프로그램 내에 코딩 되었다면 프로그램이 실행될 때 실행시간 오류가 발생

프로그램의 인터럽트

- 프로그램 실행중에 사용자가 프로그램을 잠시 중단시키거나 또한 완전히 종료시키기를 원할 경우가 있음
- 예를 들면, MS-DOS와 UNIX에서 인터럽트를 걸기 위해 <ctrl-c>를 통상 사용함
- UNIX에서 <ctrl-d> 입력 후 carriage return 신호가 파일의 끝을 알리는 전형적인 신호

입출력재지정

- MS-DOS 와 UNIX를 포함한 많은 운영체제에서 입출력을 재지정(redirect)할 수 있음

```
$ls > tmp
```

- > 기호는 운영체제가 명령의 출력을 tmp파일로 재지정하게 함

입출력재지정

```
/* dbl_out.c */
#include <stdio.h>

int main(void)
{
    char    c;
    while (scanf("%c", &c) == 1) {
        printf("%c", c);
        printf("%c", c);
    }
    return 0;
}
```

입출력재지정

- 프로그램을 컴파일하고 dbl_out 파일이란 실행 코드를 만들었다면 다음 4가지 방법으로 재지정을 이용하여 프로그램을 호출할 수 있음

```
dbl_out
dbl_out < infile
dbl_out > outfile
dbl_out < infile > outfile
```