

Chapter #1: **AN OVERVIEW OF C**

A Book on C, 3rd Ed.

Al Kelley and Ira Pohl

Addison-Wesley Publishing Company, Inc.

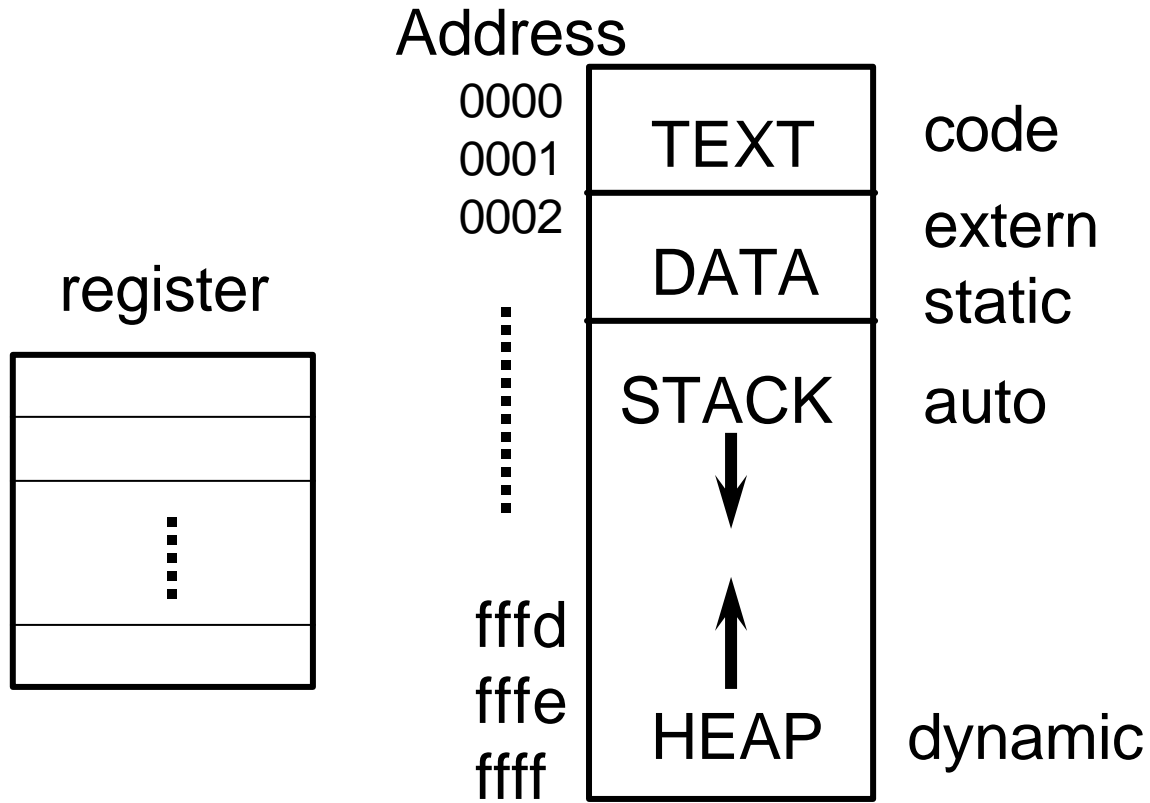
Copyright(c) 1996, 1997, SungKyunKwan University, All rights reserved.

No part of these slides may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the prior written permission of School of Electrical and Computer Engineering, SungKyunKwan University.

Machine Model for C

- Arithmetic Logic Unit
- Files
 - Disk files
 - Standard Input
 - (Keyboard)
 - Standard Output
 - (Monitor Screen)
 - Other I/O's
- Memory
 - Code memory
 - Registers
 - Data memory
 - Stack memory
 - Heap memory

Memory Model for C



Program output(1)

- The next program prints on the screen.

```
#include <stdio.h>

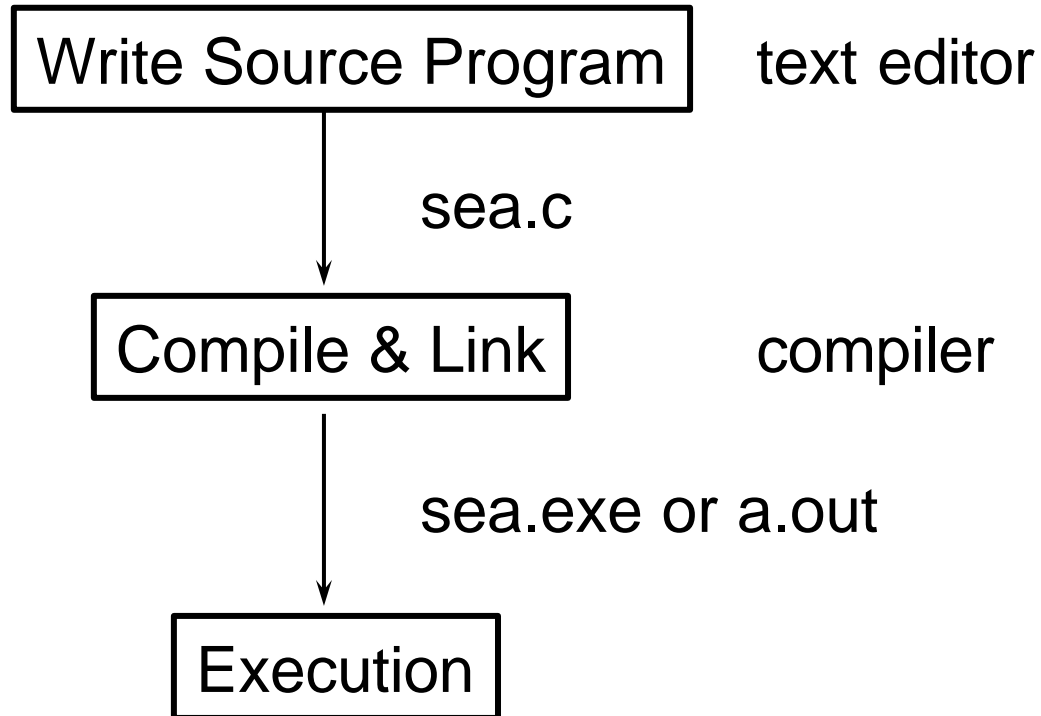
int main(void)
{
    printf("from sea to shining C\n");
    return 0;
}
```

- Using a text editor, we type this into a file whose name ends in .c.
- The choice of a file name should be mnemonic.

Program output(2)

- To compile the program, we give the command
`cc sea.c`
- The executable file a.out is created by this command.
- "a.out" executes the program and prints on the screen.
`from sea to shining C`

C Programming



Variables, Expressions, and Assignment(1)

- In C, all variables must be declared at the beginning of the program
- A variable name(identifier) consists of a sequence of letters, digits, underscore, but may not start with a digit.

```
int    miles, yards, pacific_sea;
float  kilometers;
char   c1, c2, c3;
char   1c, 2c, 3c;           /* wrong */
```

Variables, Expressions, and Assignment(2)

- The evaluation of expressions can involve conversion rules.

```
7/2          /* the int value 3 */
```

```
7.0/2       /* the value 3.5 */
```

The second expression is automatically converted to a double.

- We will write a program to convert the distance of a marathon in miles and yards to kilometers.

Variables, Expressions, and Assignment(3)

```
/* 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/1769.0);  
    printf("\nA marathon is %f kilometers.  
          \n\n", kilometers) ;  
  
    return 0 ;  
}
```

The Use of #define and #include(1)

- The C compiler has a preprocessor built into it. Lines that begin with a # are called **preprocessing directives**.

```
#define LIMIT 100
#define PI 3.14159
```

- A #define line can occur **anywhere** in a program. It affects only the lines in the file that **come after it**.
- The contents of quoted strings are never changed by the preprocessor.

```
printf("PI = %f\n", PI);
```

The Use of #define and #include(2)

- The use of symbolic constants in a program make it **more readable**

```
/* speed of light in km/sec */
```

```
#define C 299792.458
```

⇒ it will be easy to change the code at some later time.

- All the code is updated by simply changing the constant in the #define line.

The Use of #define and #include(3)

- A preprocessing directive causes a **copy** of the file when compilation occurs.

```
In myfile.h
#define PI 3.141592
```

```
In myprog.c
#include "my_file.h"
main()
{
    .....
```

```
#define PI 3.141592
main()
{
    .....
```

The Use of #define and #include(4)

- The C system provides a number of **standard header files.**

(example) `stdio.h`, `string.h`, `math.h`.

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

The use of printf() and scanf()(1)

- The function printf() is used for output and scanf() is used for input.

```
printf(" Input character : ");  
scanf("%c", &c1);
```

- The header file "stdio.h" should be included whenever the function printf() or scanf() is used

The use of printf() and scanf()(2)

- conversion specifications begins with a "%" character and ends with a conversion character.

(example) %d, %s, %c

- To print the letters abc on the screen, we could use the following statements

```
printf("abc");
```

```
printf("%s", "abc");
```

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

The use of printf() and scanf()(3)

- 'a' is the **character constant** corresponding to the lowercase letter a.
- When an argument is printed, the place where it is printed is called its **field** and the number of characters in its field is called its **field width**

```
printf("%c%3c%5c\n", 'A' , 'B' , 'C');
```

```
A__B____C
```

The use of printf() and scanf()(4)

printf()

Conversion character	How the corresponding argument is printed	
c	as a character	
d	as a decimal integer	
e	as a floating point number in scientific notation	1.2e-01
f	as a floating point number	0.12
g	in the e-format or f-format whichever is shorter as a string	
s	as a string	

The use of printf() and scanf() (5)

- The function scanf() is analogous to the function printf() but is used for input.

```
scanf ( "%d" , &x) ;
```

- The function scanf() return as an int the number of successful conversions accomplished.
- The function printf() returns as an int the number of characters printed, or a negative value in case of an error.

The use of printf() and scanf()(6)

```
scanf ( "%d" , &x ) ;
```

- Because the symbol & is the address operator, the expression &x is read "the **address of x**".
- The format %d is matched with the expression &x, causing scanf() to interpret characters in the input stream as a decimal integer and to store the result at the address of x.

The use of printf() and scanf()(7)

scanf()

Conversion character	What characters in the input stream are converted to
c	to a character
d	to a decimal integer
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

The use of printf() and scanf()(8)

```
/* The following program reads in three characters
   and some numbers, and then prints them out */
```

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

```
int main(void)
```

```
{
```

```
    char    c1,c2,c3;
```

```
    int     i;
```

```
    float   x;
```

```
    double  y;
```

```
    .....
```

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

```
    printf("\nHere is the data you typed in:\n");
```

```
    printf("%3c%3c%3c%5d%17e%17ebs\n\n",
```

```
           c1, c2, c3, i, x, y);
```

```
    .....
```

Input three characters,
an int, a float, and a double:

ABC 3 55 77.7

Here is the data you typed in:

A B C 3

5.500000e+01 7.770000e+01

The use of printf() and scanf()(9)

- When reading in numbers, scanf() will skip white space (blanks, newlines, and tabs), but when reading in a character, white space is not skipped.
- The program will not run correctly with the input **AB_C_3_55_77.7**.
- The third character is a blank, which is a perfectly good character; but then scanf() attempts to read C as a decimal integer.

Flow of control(1)

- In C, logical expressions have either the int value 1 or the int value 0.
- In C, any nonzero value is considered to represent true, and zero value is considered to represent false.
- If expression is nonzero(true), statement is executed; otherwise it is skipped.

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

Flow of control(2)

- The whole construct, even though it contains statements, is itself a single statement.

```
if(cnt==0) {  
    a = 2;  
    b = 3;  
    c = 5;  
}  
else {  
    a = -1;  
    b = -2;  
    c = -3;  
}  
printf("%d", a+b+c);
```

Flow of control(3)

- The following program illustrates the use of a while loop :

```
#include <stdio.h>

int main(void)
{
    int i = 1, sum = 0;

    while(i <= 5) {
        sum += i;
        ++i;
    }
    printf("sum = %d\n", sum);
    .....
}
```

Flow of control(4)

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

- If all three expressions are present, then this is equivalent to

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

- Typically, expr1 performs an initial assignment, expr2 performs a test, and expr3 increments a stored value.

Flow of control(5)

- Our next program illustrates the use of an if-else statement within a for loop.
- Numbers are read in one after another.

```
.....  
for(i=2; scanf("%lf", &x)==1; ++i) {  
    if(x < min) min = x;  
    else if(x > max) max = x;  
    sum += x;  
    avg = sum / i;  
}
```

.....

Functions(1)

- A C program consists of one or more functions in one or more files.
- One of functions is a main() function, where execution of the program begins. Other functions are called from within main() and from within each other.

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

subfunction1(argument)
{
    .....
}
```

Functions(2)

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

- Functions should be declared before they are used. function declarations of this type are called function prototypes.
- A function prototype tells the compiler the number and type of arguments that are to be passed to the function and the type of the value that is to be returned by the function.

Functions(3)

```
/* We illustrate the program to find minimum and maximum values. */
```

```
int main(void)  
{
```

```
.....
```

```
scanf("%d", &n); /* the number of input */
```

```
scanf("%f", &x); /* read in real number */
```

```
max = min = x;
```

```
for(i = 2; i <= n; ++i) {
```

```
    scanf("%f", &x);
```

```
    max = maximum(max, x);
```

```
    min = minimum(min, x);
```

```
}
```

```
.....
```

```
}
```

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

```
{
```

```
    if (x > y) return x;
```

```
    return y;
```

```
}
```

```
float minimum(float x, float y)
```

```
{
```

```
    if(x < y) return x;
```

```
    return y;
```

```
}
```

Functions(4)

- In C, arguments to functions are always passed "by value".
- The variables passed as arguments to functions are not changed in the calling environment.
- When *a* is passed as an argument, the expression *a* is evaluated to produce a copy of *a*. In the calling environment the variable *a* does not get changed.
- The next program illustrates this:

Functions(5)

```
/* call by value */
.....
void try_to_change_it(int);

int main(void)
{
    int a = 1;

    printf("%d\n", a); /* 1 is printed */
    try_to_change_it(a);
    printf("%d\n", a); /* 1 is printed */
    ...
}

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

Arrays, Strings and Pointers(1)

- In C, a string is an array of characters, and an array name by itself is a pointer.

Arrays

- Arrays are used when many variables, all of the same type, are desired.

```
int a[3]
```

- The elements of the array are of type int and are accessed as a[0], a[1], and a[2]. The index, or subscript, of an array always starts at 0.

Arrays, Strings and Pointers(2)

```
/* 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;
        }
    }
}
.....
```

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

Arrays, Strings and Pointers(3)

- A bubble sort is used in the program to sort the scores.
- If the elements being compared are out of order, their values are interchanged. Here, this interchange is accomplished by the code

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

Arrays, Strings and Pointers(4)

Strings

- In C, a string is an array of characters.
- In this section, in addition to illustrating the use of strings, we want to introduce the use of getchar() and putchar().
- Our next program stores a line typed in by user in an array of characters(a string), and then prints the line backwards on the screen.

Arrays, Strings and Pointers(5)

```
.....
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 !");
.....
```

Arrays, Strings and Pointers(6)

- If we run the program and enter the name *Alice B Carole*:

```
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!
```

Arrays, Strings and Pointers(7)

Pointers

- A pointer is an address of an object in memory.
- An array name is itself a pointer.
- The following program is designed to illustrate some of these relationships.

Arrays, Strings and Pointers(8)

```
/* 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);
.....
```

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

Arrays, Strings and Pointers(9)

- In C, arrays, strings, and pointers are closely related.

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

- 'p' is a variable pointer whereas 's' is a constant pointer that points to s[0].
- s[i] and *(s + i) are equivalent.
- In a similar fashion, p[i] and *(p + i) are equivalent.

Files(1)

```
#include <stdio.h>

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

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

- The second line in the body of main() declares `ifp` to be a pointer to `FILE`. the type `FILE` is defined in `stdio.h` as a particular structure.

Files(2)

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

- The function `fopen()` takes two strings as arguments, and it returns a pointer to `FILE`.
- The first argument is the name of the file, and the second argument is the mode in which the file is to be opened.
- The function `fopen()` is in the standard library, and its function prototype is in `stdio.h`.
- If for some reason a file cannot be accessed, the pointer value `NULL` is returned by `fopen()`.

Files(3)

- Let us describe how command line arguments can be accessed from within a program

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

- The parameter *argc* stands for "argument count". The parameter *argv* stands for "argument variable". It is an array of pointers to char.
- we will write a program that counts the occurrences of just uppercase letters.

Files(4)

```
/* count uppercase letters */
.....
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);
    .....
}
```

Files(5)

- The program should read file *chapter1* and *data1* as command line arguments.

```
cnt_letters chapter1 data1
```

- This is what we find in the file *data1*:

```
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
```

Operating system considerations(1)

- The precise steps that have to be followed to create a file containing C code and to compile and execute it depend on three things:

the operating system

the text editor

the compiler

Operating system considerations(2)

* Steps to be followed in writing and running a C program

1. Using an editor, create a text file, say `pgm.c`, that contains a C program. The name of the file must end with `.c`.

2. Compile the program.

```
cc pgm.c
```

3. Execute the program

```
a.out
```

Operating system considerations(3)

```
cc -o sea sea.c
```

- In UNIX, it is common practice to give the executable file the same name as the corresponding source file, except to drop the .c suffix.
- Syntax errors are caught by the compiler, whereas run-time errors manifest themselves only during program execution.
- For example, if an attempt to divide by zero is encoded into a program, a run-time error may occur when the program is executed.

Operating system considerations(4)

- * **Let us now consider an MS-DOS environment.**
- Some C systems, such as Turbo C, have both a command line environment and an integrated environment.
- In MS-DOS, the executable output produced by a C compiler is written to a file having the same name as the source file, but with the extension .exe instead of .C.

```
tcc sea.c /* compile */  
sea.exe or equivalently sea /* execute */
```

Operating system considerations(5)

- When running a program, the user may interrupt, or kill, the program.
- For example, the program may be in an infinite loop. In MS-DOS and in UNIX, a control-c is commonly used to effect an interrupt.
- In UNIX, a carriage return followed by a control-d is the typical way to effect an end-of-file signal. In MS-DOS a control-z must be typed instead.

Operating system considerations(6)

- Many operating systems, including MS-DOS and UNIX, can redirect the input and the output.

```
ls > tmp
```

- The symbol `>` causes the operating system to redirect the output of the command to the file *tmp*.

Operating system considerations(7)

```
/* The program reads characters from the
   standard input file, and writes each
   character twice to standard output file */

#include <stdio.h>

int main(void)
{
    char c ;

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

Operating system considerations(8)

- We compile the program and put the executable code in the file *dbl_out*.
- Using redirection, we can invoke the program in four ways.

```
dbl_out
```

```
dbl_out < infile
```

```
dbl_out > outfile
```

```
dbl_out < infile > outfile
```