C Fundamentals & Formatted Input/Output

adopted from KNK C Programming : A Modern Approach

Program: Printing a Pun – pun.c

- The file name doesn't matter, but the `.c` extension is often required.
  - for example: pun.c

```
#include <stdio.h>

int main(void)
{
  printf("To C, or not to C: that is the question.\n");
  return 0;
}
```

- to compile

```
% cc -o pun pun.c
```

or

```
% gcc -o pun pun.c
```

- to run

```
% ./pun
```

Program: Printing a Pun Revised – pun.c

- `printf` doesn't automatically advance to the next output line when it finishes printing.

```
#include <stdio.h>

int main(void)
{
  printf("To C");
  printf(", or not to C:");
  printf(" that i");
  printf(" s the question. ");
  printf(" \n");
  return 0;
}
```

Program: Computing the Dimensional Weight of a Box

- use `dweight.c`

- Shipping companies often charge extra for boxes that are large but very light, basing the fee on volume instead of weight.
- The usual method to compute the "dimensional weight" is to divide the volume by 166 (the allowable number of cubic inches per pound).
- The `dweight.c` program computes the dimensional weight of a particular box:

```
Dimensions: 12x10x8
Volume (cubic inches): 960
Dimensional weight (pounds): 6
```

Program: Computing the Dimensional Weight of a Box

- Division is represented by / in C, so the obvious way to compute the dimensional weight would be

```
weight = volume / 166;
```

- In C, however, when one integer is divided by another, the answer is "truncated": all digits after the decimal point are lost.
  - The volume of a 12" × 10" × 8" box will be 960 cubic inches.
  - Dividing by 166 gives 5 instead of 5.783.

- One solution is to add 165 to the volume before dividing by 166:

```
weight = (volume + 165) / 166;
```

- A volume of 166 would give a weight of 331/166, or 1, while a volume of 167 would yield 332/166, or 2.

dweight.c

```
/* Computes the dimensional weight of a 12" x 10" x 8" box */

#include <stdio.h>

int main(void)
{
  int height, length, width, volume, weight;

  height = 8;
  length = 12;
  width = 10;
  volume = height * length * width;
  weight = (volume + 165) / 166;

  printf("Dimensions: %dx%dx%d\n", length, width, height);
  printf("Volume (cubic inches): %d\n", volume);
  printf("Dimensional weight (pounds): %d\n", weight);

  return 0;
}
```

Program: Computing the Dimensional Weight of a Box (Revisited)

- `dweight2.c` is an improved version of the dimensional weight program in which the user enters the dimensions.
- Each call of `scanf` is immediately preceded by a call of `printf` that displays a ***prompt.***

dweight2.c

```
/* Computes the dimensional weight of a box from input
provided by the user */

#include <stdio.h>

int main(void)
{
  int height, length, width, volume, weight;

  printf("Enter height of box: ");
  scanf("%d", &height);
  printf("Enter length of box: ");
  scanf("%d", &length);
  printf("Enter width of box: ");
  scanf("%d", &width);
  volume = height * length * width;
  weight = (volume + 165) / 166;

  printf("Volume (cubic inches): %d\n", volume);
  printf("Dimensional weight (pounds): %d\n", weight);

  return 0;
}
```

Program: Computing the Dimensional Weight of a Box (Revisited)

- Sample output of program:

```
Enter height of box: 8
Enter length of box: 12
Enter width of box: 10
Volume (cubic inches): 960
Dimensional weight (pounds): 6
```

- Note that a prompt shouldn't end with a new-line character.

Defining Names for Constants

- `dweight.c` and `dweight2.c` rely on the constant 166, whose meaning may not be clear to someone reading the program.
- Using a feature known as ***macro definition,*** we can name this constant:

```
#define INCHES_PER_POUND 166
```

- TODO:
  - Change the code to use the macro definition

Program: Converting from Fahrenheit to Celsius

- The `celsius.c` program prompts the user to enter a Fahrenheit temperature; it then prints the equivalent Celsius temperature.
- Sample program output:

```
Enter Fahrenheit temperature: 212
Celsius equivalent: 100.0
```

- The program will allow temperatures that aren't integers.

- Defining `SCALE_FACTOR` to be `(5.0f / 9.0f)` instead of `(5 / 9)` is important.
- Note the use of `%.1f` to display `celsius` with just one digit after the decimal point.

celsius.c

```
/* Converts a Fahrenheit temperature to Celsius */

#include <stdio.h>

#define FREEZING_PT 32.0f
#define SCALE_FACTOR (5.0f / 9.0f)

int main(void)
{
  float fahrenheit, celsius;

  printf("Enter Fahrenheit temperature: ");
  scanf("%f", &fahrenheit);

  celsius = (fahrenheit - FREEZING_PT) * SCALE_FACTOR;

  printf("Celsius equivalent: %.1f\n", celsius);

  return 0;
}
```

Layout of a C Program (1/2)

- The amount of space between tokens usually isn't critical.
- At one extreme, tokens can be crammed together with no space between them, except where this would cause two tokens to merge:

```
/* Converts a Fahrenheit temperature to Celsius */
#include <stdio.h>
#define FREEZING_PT 32.0f
#define SCALE_FACTOR (5.0f/9.0f)
int main(void){float fahrenheit,celsius;printf(
"Enter Fahrenheit temperature: ");scanf("%f", &fahrenheit);
celsius=(fahrenheit-FREEZING_PT)*SCALE_FACTOR;
printf("Celsius equivalent: %.1f\n", celsius);return 0;}
```

- adding spaces and blank lines to a program can make it easier to read and understand.

Layout of a C Program (2/2)

- Although extra spaces can be added between tokens, it's not possible to add space within a token without changing the meaning of the program or causing an error.

```
fl oat fahrenheit, celsius;          /*** WRONG ***/

fl
      oat fahrenheit, celsius;       /*** WRONG ***/
```

- Putting a space inside a string literal is allowed, although it changes the meaning of the string.
- Putting a new-line character in a string (splitting the string over two lines) is illegal:

```
/*** WRONG ***/
printf("To C, or not to C:
       that is the question.\n");
```

Program: Adding Fractions

- The `addfrac.c` program prompts the user to enter two fractions and then displays their sum.
- Sample program output:

```
Enter first fraction: 5/6
Enter second fraction: 3/4
The sum is 38/24
```

Program: Adding Fractions

```
/* Adds two fractions */

#include <stdio.h>

int main(void)
{
  int num1, denom1, num2, denom2, result_num, result_denom;

  printf("Enter first fraction: ");
  scanf("%d/%d", &num1, &denom1);

  printf("Enter second fraction: ");
  scanf("%d/%d", &num2, &denom2);

  result_num = num1 * denom2 + num2 *denom1;
  result_denom = denom1 * denom2;
  printf("The sum is %d/%d\n",result_num, result_denom)

  return 0;
}
```

Program: Using `printf` to Format Numbers

- The `tprintf.c` program uses `printf` to display integers and floating-point numbers in various formats.

```
/* Prints int and float values in various formats */

#include <stdio.h>

int main(void)
{
  int i;
  float x;

  i = 40;
  x = 839.21f;

  printf("|%d|%5d|%-5d|%5.3d|\n", i, i, i, i);
  printf("|%10.3f|%10.3e|%-10g|\n", x, x, x);

  return 0;
}
```

Use `printf` to format the following string

- 2 digits for Major Number
- 2 digits for Minor Number
- 1 digit for cpu number
- 6 digit for sequence number
- nano second precision for timestamp
- 6 digit for process ID
- 1 character for command
- 1 character for I/O operation
- 10 characters for content

```
8,32 3 1 0.000000000 2208 Q R 0 + 2 [dd]
8,32 3 2 0.000002113 2208 G R 0 + 2 [dd]
```

How `scanf` Works (1/4)

- As it searches for a number, `scanf` ignores ***white-space characters***
  - space, horizontal and vertical tab, form-feed, and new-line
- A call of `scanf` that reads four numbers:

```
scanf("%d%d%f%f", &i, &j, &x, &y);
```

- The numbers can be on one line or spread over several lines:

```
  1
-20   .3
   -4.0e3
```

```
••1¤-20•••.3¤•••-4.0e3¤
ssrsrrrsssrrssssrrrrrr
```

(`s` = skipped; `r` = read)

- `scanf` "peeks" at the final new-line without reading it.

How scanf Works (2/4)

```
#include <stdio.h>
int main()
{
    int i, j;
    float x, y;
    scanf("%d%d%f%f", &i, &j, &x, &y);
    printf("i:%d\tj:%d\tx:%f\ty:%f\n", i, j, x, y);
    return;
}
```

1

```
  1
-20   .3
   -4.0e3
```

2

```
1-20.3-4.0e3¤
```

How `scanf` Works (3/4)

```
#include <stdio.h>
int main()
{
    int i, j;
    scanf("%d/%d", &i, &j);
    printf("i:%d\tj:%d\n", i, j);
    return;
}
```

1 `38/ 28`

2 `38 /28`

How scanf Works (4/4)

```
#include <stdio.h>
int main()
{
    int i, j;
    scanf(”  %d/hello/%d", &i, &j);
    printf(“i:%d\tj:%d\n”, i, j);
    return;
}
```

1 38/ 28

2 38 /28

3 38 / 28

4 38/hel lo/ 28

5 38/hello/ 28