Technical Informatics I

Variables and standard functions

by
Dr. Fatimah
Faculty of Mechanical Engineering
fatimahd@ump.edu.my

Variables and standard functions

- Aims
  - Introduce students to variable types and variable declaration
  - Introduce students input output formatting (`printf` and `scanf`)
  - Introduce students to Escape Sequences

- Expected Outcomes
  - Students are able to construct simple C programs that can display formatted data using `printf()`
  - Students are able to construct simple C programs that can takie in user input using `scanf()`

- References
  - Harry H. Cheng, 2010. C for Engineers and Scientists: An Interpretive Approach, McGraw Hill

Content

- Introduction to Variable Types
- Variable Declarations
- Function printf()
  - With arguments
  - Without arguments
  - Escape Sequences
- Function scanf()
  - With arguments
  - Without arguments
- Example
- Conclusion

Introduction to Variable Types

- A variable refers to the storage area to be manipulated and referenced by programs.
- Variable **type**
  - Each variable in C has a specific ***type***
  - Determines the size and layout of the variable's memory
  - Determines the set of operations that can be applied to the variable and the range of values that can be stored within that memory

Introduction to Variable Types

- There are lots of variable types in C
- To keep things simple but yet enough to write a sufficiently useful C program, we are only going to look at 4 variable types
- These variable types should be able to give you a good head start for you to further delve into more sophisticated variable types

Variable Types

- Integer Type

Type	Storage size	Value range
char	1 byte	-128 to 127 or 0 to 255
int	2 or 4 bytes	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647

- Floating-point Types

Type	Storage size	Value range	Precision
float	4 bytes	1.2E-38 to 3.4E+38	6 decimal places
double	8 bytes	2.3E-308 to 1.7E+308	15 decimal places

Variable Declaration

- All variables MUST be declared BEFORE the variable is called/used
- All type declarations follow a standard format:

```
variable_type variable_name;
```

- Where **variable_type** represents one of the C data type (see previous slide: int/char/float/double)
- Where **variable_name** would be replaced by the programmer with a name for the variable

Variable Declaration

- Example of declaration:

```
#include <stdio.h>
int main(){
    int var1;
    char c2;
    float p3;
    double number5;
return 0;
}
```

Rules for Variable Declaration

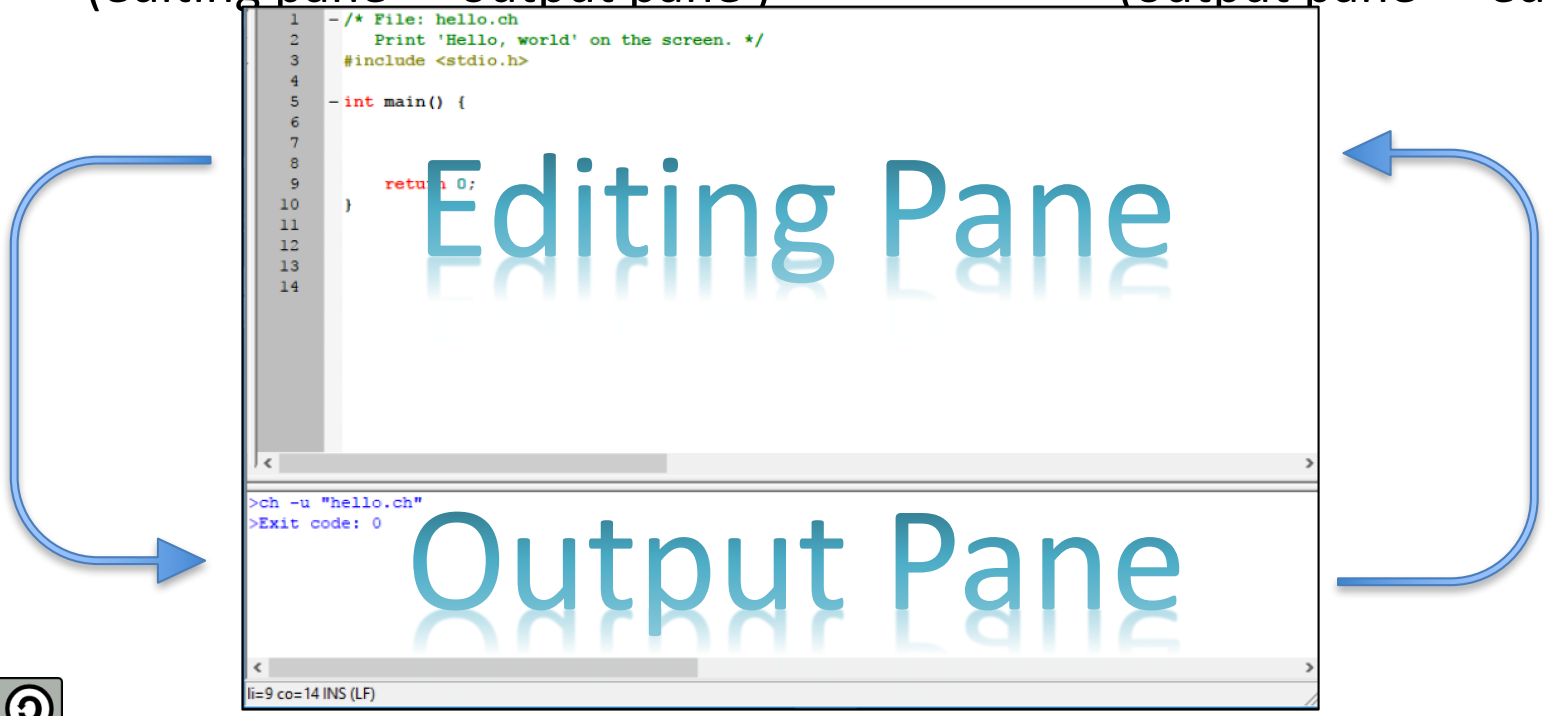
- According to C standards there are a few restriction on variable names:
  - Must be at least 1 character long
  - Must be less than some maximum character length (32 on some systems, 256 on others)
  - Must start with a letter
  - May be composed of letters, numbers and/or underscore character

Rules for Variable Declaration

- Must **NOT** contain spaces
- Must not be equal to reserved words such as `int, float, char` etc. Any keywords used for C itself is off-limits
- Case sensitive:
  - Thus “MYNAME”, “myname”, “Myname”, “MyName” and “mYnaME” are all different variables

Functions `printf()` and `scanf()`

- `printf()`
  - displays output
  - (editing pane -> output pane)
- `scanf()`
  - take input from users
  - (output pane -> editing pane)



Functions `printf()` and `scanf()`

- Precise formatted output is accomplished using the output function: **`printf`**
- Precise formatting input is accomplished using the input function: **`scanf`**

Format specifiers

Type	Format specifier: printf()	Format specifier: scanf()
int	%d	%d
char	%c	%c
float	%f	%f
double	%f	%lf

Functions `printf()`

The **printf** function has following form

```
printf(format-control-string,
       arguments);
```

- **Format-control-string**: Specification to describe output format. Enclosed in quotation marks("…"). To call variable(s), the format specifier begins with a percent sign (%) followed by a conversion specifier listed in the table in the following slide
- **Arguments**: Variable that corresponds to the format specifier called in the format-control-string.

Functions `printf()`

There are 2 possible syntax for printf:

SYNTAX:

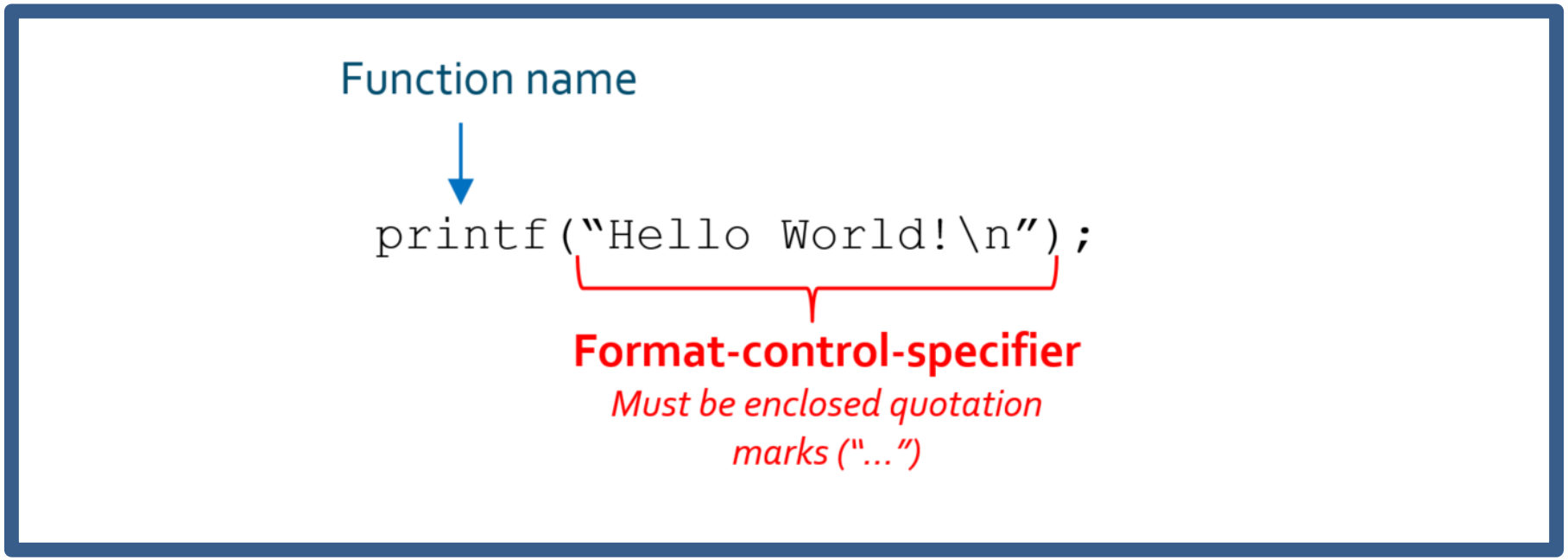
```
1. printf(format-control-string);
2. printf(format-control-string, variable-arguments);
```

EXAMPLES:

```
1. printf("Hello World!\n");
2. printf("That weight is %f kilograms\n",weight);
```

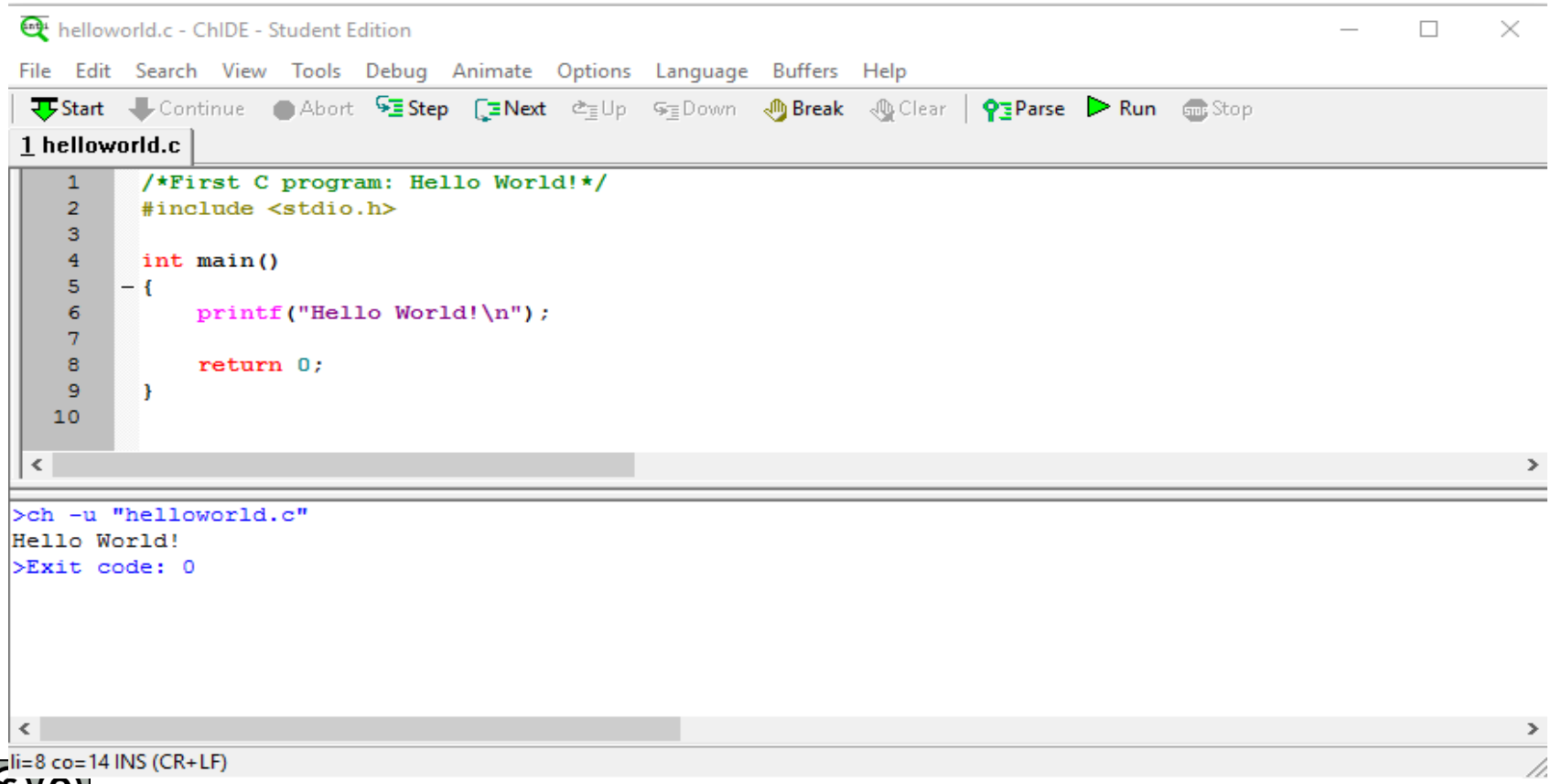
Functions `printf()`

```
printf(format-control-string);
```



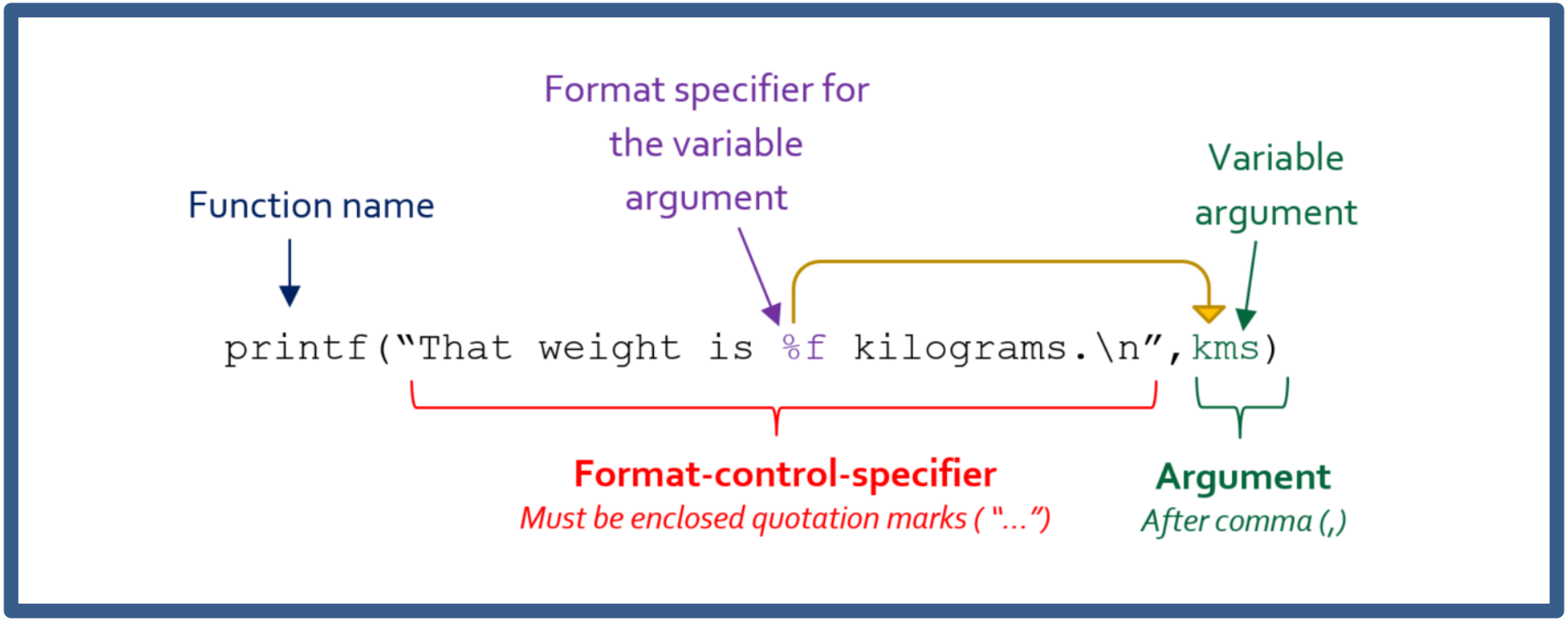
Functions `printf()`

- **Example 1:** `printf(format-control-string);`



Functions `printf()`

```
printf(format-control-string, variable-arguments);
```



Functions `printf()`

- **Example 2:**

```
printf(format-control-string, variable-arguments);
```

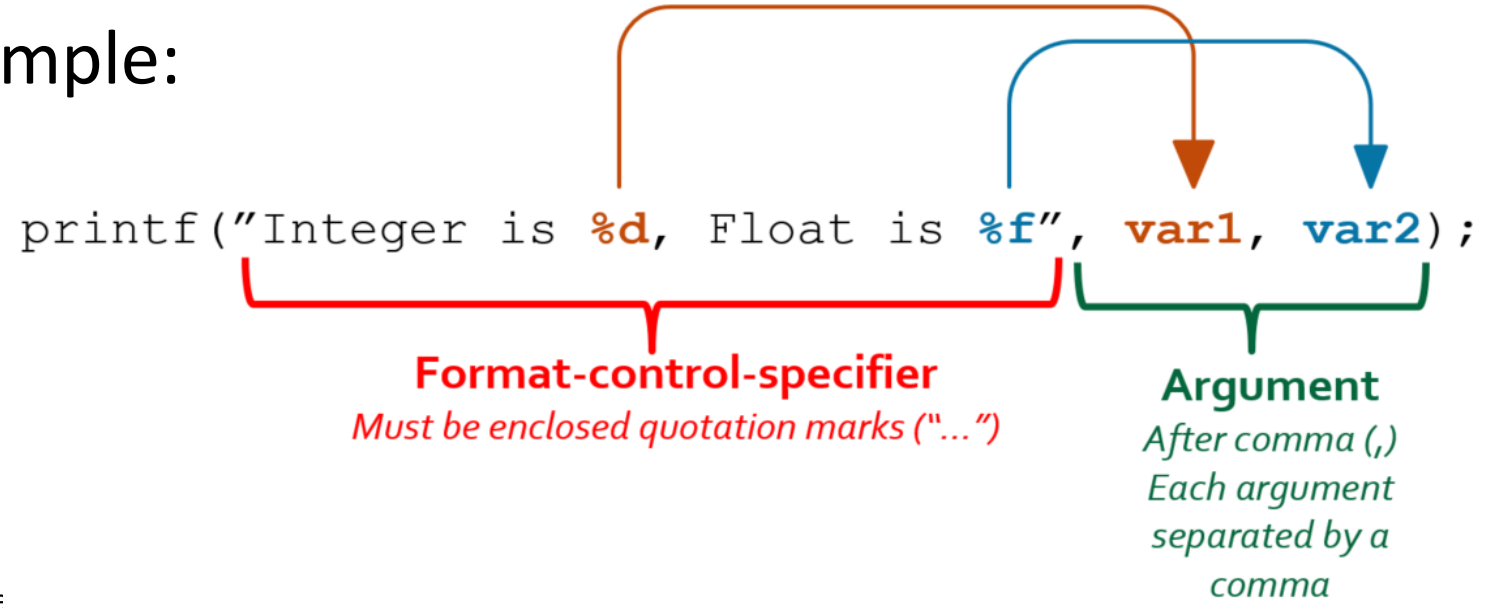
```
1  /*printf example*/
2  int main(){
3      double kms=300;
4      printf("That equals %f kilometers.\n",kms);
5
6      return 0;
7  }
8
```

```
>ch -u "printwithargseg1.c"
That equals 300.000000 kilometers.
>Exit code: 0
```

Functions `printf()`

Printing Multiple Numerical Values in a Single Printing Statement

- Use multiple format specifiers.
- Each format specifier corresponds to an argument.
- Example:



Functions `printf()`

- **Example 3:** Printing out multiple variables

```c
/* Example of printf function with multiple variable arguments*/
int main()
{
    int var1=9;
    float var2=12.3454;

    printf("Int is %d, Float is %f\n", var1, var2);

    return 0;
}
```

```
>ch -u "printwithargmult.c"
Int is 9, Float is 12.345400
>Exit code: 0
```

Functions `printf()`

Precision of Floating-Point Numbers

- The precision of a floating-point number: number of digits after the decimal point.
- For example, the format “%.8f” specifies the precision with 8 digits after the decimal point.

```
Example 1: printf(”%.3f”, 67.893712)
Output: 12.894

Example 2: printf(”%.21f”, 0.8)
Output: 0.800000000000000011100
```

Functions `printf()`

Precision of Floating-Point Numbers

Precision	Description
%d	prints out a decimal integer)
%3d	prints out a decimal integer with a width of *at least* 3
%f	prints out a floating point)
%4f	prints out as a floating point with a width of *at least* 4
%.7f	prints out a floating point with a precision of 7 characters after the decimal point
%5.1f	prints out a floating point at least 5 wide and a precision of 1)

Functions `printf()`

Precision of Floating-Point Numbers

```c
/*Ali Yusuf bin Muhammad Ali MA12345*/
/*File: lab2ex3.c. Prints out floating point for different precisions and widths*/

#include <stdio.h>
int main() {
    /*Variable declaration*/
    double testVariable = 3645.901234671;

    printf(" Precision(dp: decimal point) & width(spaces)   |    Output\n");
    printf("-------------------------------------------------------------------------\n");
    printf(" Precision: none & Width: none                  |    %f \n", testVariable);
    printf(" Precision: 2dp  & Width: none                  |    %.2f \n", testVariable);
    printf(" Precision: 4dp  & Width: none                  |    %.4f \n", testVariable);
    printf(" Precision: 6dp  & Width: none                  |    %.6f \n", testVariable);
    printf(" Precision: 2dp  & Width: 5                     |    %5.2f \n", testVariable);
    printf(" Precision: 2dp  & Width: 6                     |    %6.2f \n", testVariable);
    printf(" Precision: 2dp  & Width: 7                     |    %7.2f \n", testVariable);
    printf(" Precision: 2dp  & Width: 8                     |    %8.2f \n", testVariable);
    printf(" Precision: 2dp  & Width: 9                     |    %9.2f \n", testVariable);
    printf(" Precision: 2dp  & Width: 10                    |    %10.2f \n", testVariable);
    printf(" Precision: none & Width: 10                    |    %10f \n", testVariable);
    printf(" Precision: none & Width: 15                    |    %15f \n", testVariable);
    printf(" Precision: none & Width: 20                    |    %20f \n", testVariable);
    printf(" Precision: 7dp & Width: 10                     |    %10.7f \n", testVariable);
    printf(" Precision: 7dp & Width: 15                     |    %15.7f \n", testVariable);
    printf(" Precision: 7dp & Width: 20                     |    %20.7f \n", testVariable);

    return 0;
}
```

Functions `printf()`

Precision of Floating-Point Numbers

```
>ch -u "lab2ex3.c"
 Precision(dp: decimal point) & width(spaces)   |    Output
--------------------------------------------------------------------------
 Precision: none & Width: none                  |    3645.901235
 Precision: 2dp  & Width: none                  |    3645.90
 Precision: 4dp  & Width: none                  |    3645.9012
 Precision: 6dp  & Width: none                  |    3645.901235
 Precision: 2dp  & Width: 5                     |    3645.90
 Precision: 2dp  & Width: 6                     |    3645.90
 Precision: 2dp  & Width: 7                     |    3645.90
 Precision: 2dp  & Width: 8                     |     3645.90
 Precision: 2dp  & Width: 9                     |      3645.90
 Precision: 2dp  & Width: 10                    |       3645.90
 Precision: none & Width: 10                    |    3645.901235
 Precision: none & Width: 15                    |        3645.901235
 Precision: none & Width: 20                    |             3645.901235
 Precision: 7dp & Width: 10                     |    3645.9012347
 Precision: 7dp & Width: 15                     |       3645.9012347
 Precision: 7dp & Width: 20                     |            3645.9012347
>Exit code: 0
```

Functions `printf()`

Example 4: Precision of Floating-Point Numbers

```
1  /*Example for floating point numbers*/
2  #include<stdio.h>
3  int main(){
4      double a=12.567893746;
5      printf("a=%.2f\n", a);
6      printf("a=%10.3f\n", a);
7
8      return 0;
9  }
```

```
>ch -u "eg3.c"
a=12.57
a=    12.568
>Exit code: 0
```

Escape sequences

Escape sequence	Character
\n	New line
\t	Tab
\v	Vertical tab
\\	Backslash
\?	Question Mark
\’	Single Quote
\”	Double quote

Functions `printf()`

- For more examples and tutorials for printf(), see:

http://www.codingunit.com/printf-format-specifiers-format-conversions-and-formatted-output

Functions `scanf()`

The `scanf()` function has following form

```
scanf(format-control-string, arguments );
```

- **Format-control-string**: Using specifications to describe input format. Each specification begins with a percent sign(%), ends with conversion specifier and is enclosed in quotation marks.
- **Arguments**: Variable name whose values are to be saved which is preceded by an ampersand (&).

Functions `scanf()`

- Recall format specifiers:

Type	Format specifier: printf()	Format specifier: scanf()
int	%d	%d
char	%c	%c
float	%f	%f
double	%f	%lf

Functions `scanf()`

- Example 1:

```
#include <stdio.h>
int main(){
      int i;
       char c;
       float f;
       double d;
       scanf("%d", &i);
       scanf("%c", &c);
       scanf("%f", &f);
       scanf("%lf",&d);
     return 0;
}
```

Functions `scanf()`

- Example 2: printf() and scanf() combined

```c
/* Example for printf and scanf */
#include <stdio.h>

int main() {
    int num;
    double d;

    printf("Please input an integer and one floating-point number\n");
    scanf("%d",&num);
    scanf("%lf", &d);

    printf("Your input values are %d and %f\n", num, d);
    return 0;
}
```

```
>ch -u "eg4.c"
Please input an integer and one floating-point number
1
2
Your input values are 1 and 2.000000
>Exit code: 0
```

Applications: `printf()` and `scanf()`

Example of application

Write a C program that calculates the acceleration described by the function:

$$a(t)=(p(t) - \mu mg)/m$$

The acceleration is dependent on the input values for **μ**, **m**, and the external force, p:

$$p(t) = 4(t-3)+20 \qquad \text{when } t >= 0$$

Adapted from (Cheng, 2010)

Applications: `printf()` and `scanf()`

```
/*This code calculates acceleration*/
#include <stdio.h>
#define M_G   9.81
int main() {
    double a, mu, m, p, t;    /*Variable Declaration*/

    printf("Please enter value for mass in kilogram\n");
    scanf("%lf", &m);
    printf("mass is %lf (kg)====\n\n", m);

    printf("Please enter value for friction coefficient\n");
    scanf("%lf", &mu);
    printf("friction coefficient is %lf\n\n", mu);

    printf("Please enter value for time in second\n");
    scanf("%lf", &t);
    printf("time is %lf (s)\n\n", t);

    p = 4*(t-3)+20;
    a = (p-mu*m*M_G)/m;
    printf("Acceleration a = %f (m/s^2)\n", a);
    return 0;
}
```

Adapted from (Cheng, 2010)

Applications: `printf()` and `scanf()`

```
/*This code calculates acceleration*/
#include <stdio.h>
#define M_G    9.81
int main() {
    double a, mu, m, p, t;
    printf("Please enter value for mass in kilogram\n");
    scanf("%lf", &m);
    printf("mass is %lf (kg)====\n\n", m);
    printf("Please enter value for friction coefficient\n");
    scanf("%lf", &mu);
    printf("friction coefficient is %lf\n\n", mu);
    printf("Please enter value for time in second\n");
    scanf("%lf", &t);
    printf("time is %lf (s)\n\n", t);
    p = 4*(t-3)+20;
    a = (p-mu*m*M_G)/m;
    printf("Acceleration a = %f (m/s^2)\n", a);
    return 0;
}
```

Homework:
Figure out what #define does

Adapted from (Cheng, 2010)

Conclusion

- Conclusion #1
  - Variable Declaration: `variable_type variable_name;`
- Conclusion #2
  - printf():
    1. `printf(format-control-string);`
    2. `printf(format-control-string, variable-arguments);`
  - scanf()
    1. `scanf(format-control-string, arguments );`
- Conclusion #3

Type	Format specifier: printf()	Format specifier: scanf()
int	%d	%d
char	%c	%c
float	%f	%f
double	%f	%lf

Technical Informatics I

Lecture 2

Dr Fatimah