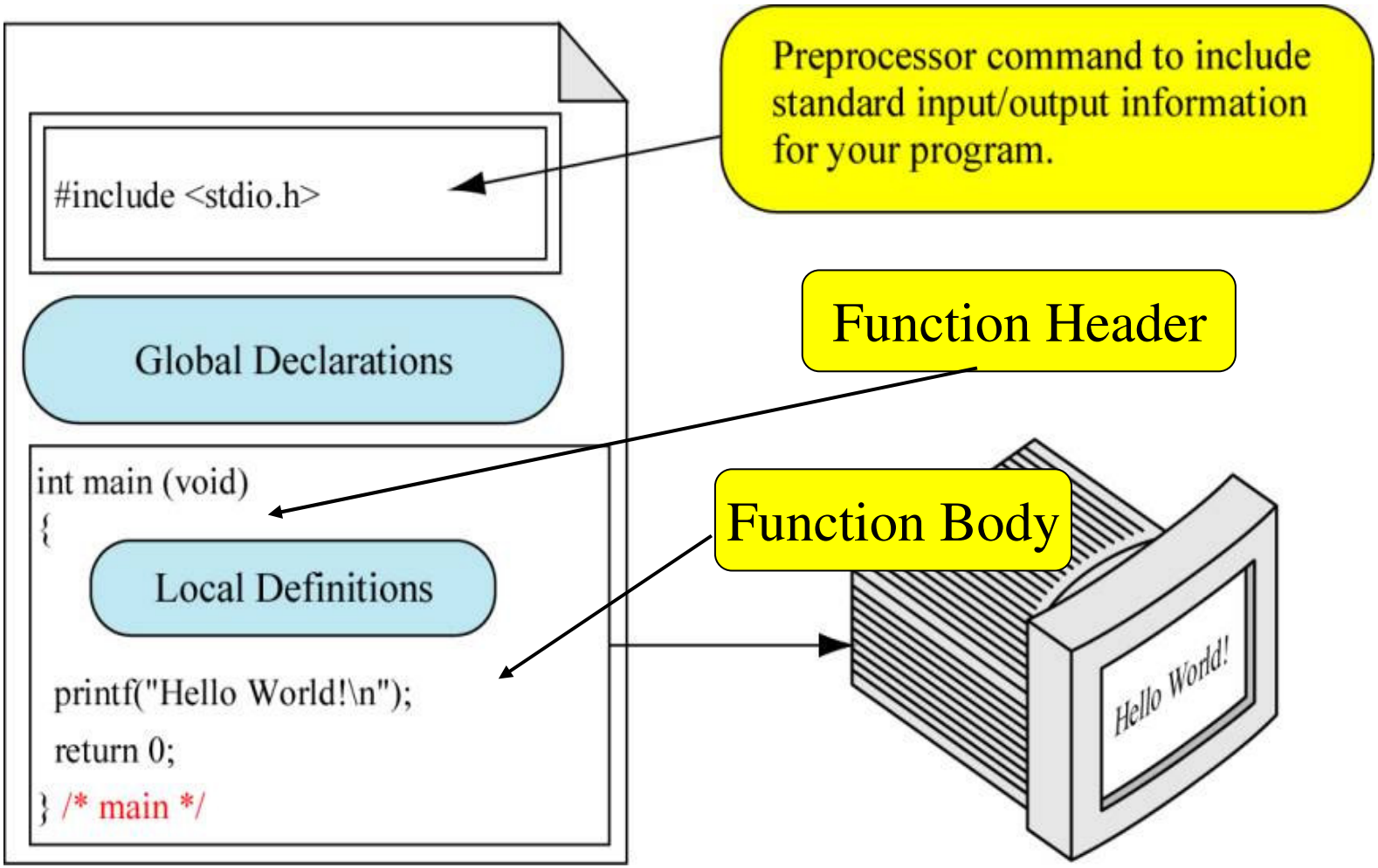
Structured Programming

Dr. Mohamed Khedr

Lecture 2

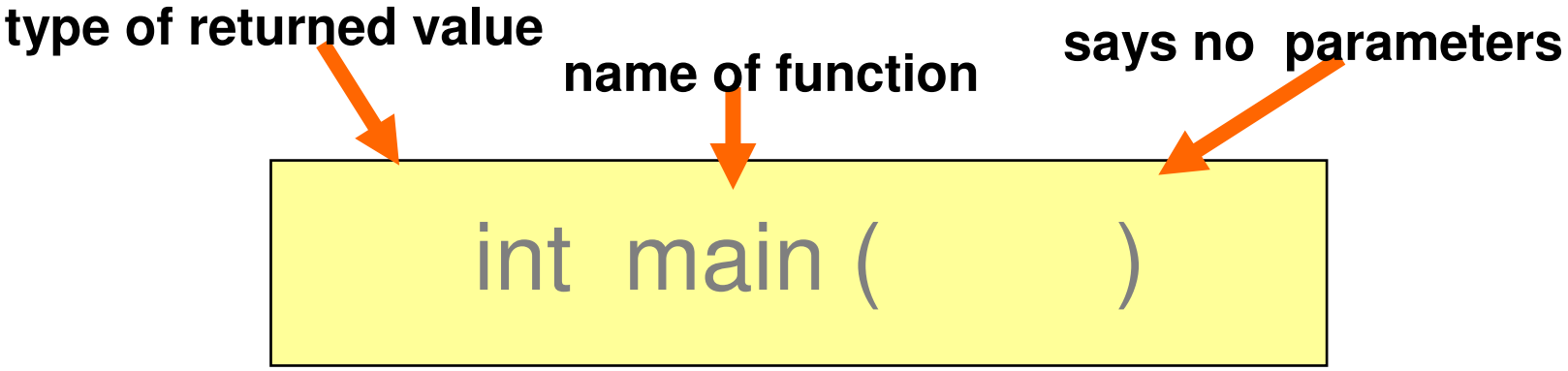
http://webmail.aast.edu/~khedr

C Program structure
A Sample C Program



C Language Elements

The Function Header



- **A C program is a collection of one or more "functions" (parts)**
- **There must be a function called main( )**
- **A function body has two parts declaration and executable statements.**
- **Execution always begins with the first statement in function main( )**
- **Any other functions in the program are subprograms and are not executed until they are called**
- **printf(" Hello, World ");**
  This statement means: Display the words Hello, World on the screen
- **return (0);**
  This statement STOPs the program and returns Zero to the O.S.

```
1  /* Fig. 2.1: fig02_01.c
2     A first program in C */
3  #include <stdio.h>
4
5  /* function main begins program execution */
6  int main( void )
7  {
8     printf( "Welcome to C!\n" );
9
10    return 0; /* indicate that program ended successfully */
11
12 } /* end function main */
```

```
Welcome to C!
```

/* and */ indicate comments – ignored by compiler

#include directive tells C to load a particular file

Left brace declares beginning of main function

Statement tells C to perform an action

return statement ends the function

Right brace declares end of main function

Outline

- fig02_01.c

Escape sequence	Description
\n	Newline. Position the cursor at the beginning of the next line.
\t	Horizontal tab. Move the cursor to the next tab stop.
\a	Alert. Sound the system bell.
\\	Backslash. Insert a backslash character in a string.
\"	Double quote. Insert a double-quote character in a string.

C Language Elements
Words

- **Reserved Word**
  - **A word that has special meaning in C**
- **Standard Identifiers**
  - **A word having special meaning but one that a programmer may redefine (but redefinition is not recommended)**
- **User Defined Identifiers**
  - **An identifier must consist only of letters, digits, and underscores.**
  - **An identifier cannot begin with a digit**
  - **A C reserved word cannot be used as an identifier.**
  - **An identifier defined in a C standard library should not be redefined.**
  - **C is a case-sensitive language. The names Pressure, pressure, and PRESSURE are viewed by the compiler as different identifier.**

C Language Elements
Invalid/Valid identifiers

- **Invalid identifiers**

  1Letter begins with a number
  double reserved word
  int reserved word
  TWO*FOUR character * not allowed
  joe's character ' not allowed
  age# character # not allowed
  Age-of-cat character – is not underscore character (_)

- **Valid Identifiers**

  Age_of_person
  taxRateY2k
  PrintHeading

C Language Elements
Words in the second example

- **Reserved Word**
  - **int**
  - **void**
  - **Double**
  - **return**
- **Standard Identifiers**
  - **Printf**
  - **scanf**
- **User Defined Identifiers**
  - **KMS_PER_MILE**
  - **miles**
  - **kms**

Keywords

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Fig. 2.15 | **C's keywords.**

Standard Data Type in C

- **Integral Types**
  - represent whole numbers and their negatives
  - declared as int, short, or long
  - Int sample values 4578, -4578, 0
- **Floating Types**
  - represent real numbers with a decimal point
  - declared as float, or double
  - Float sample values 95.274 95.0 0.265
- **Character Types**
  - represent single characters
  - declared as char
  - Char sample values B d 4 ? *

What about words and sentences ?

- **a String is a sequence of characters enclosed in double quotes**
- **sample values**

  “Hello” “Year 2004” “1234”

- **Not in Standard C, We’ll cover it later**

Variables

```
int x, y, z;
```

- Variables declared at beginning of a function
  - Before any executable statements
- Number of bytes for each data type
  - char = 1 (-128 to 127)
  - short = 2 (-32768 to 32767)
  - int & long = 4 (-2,147,483,648 to +2,147,483,647)
  - float = 4 ($\pm \sim 10^{-44}$ to $\sim 10^{38}$)
  - double = 8 ($\pm \sim 10^{-323}$ to $\sim 10^{308}$)

Variable Names

- Restrictions
  - Made up of letters & digits
  - 1st character must be a letter
  - Underscore ("_") counts as a letter
    - Often used in library routines
  - Case sensitive
    - APPLE & apple are different variables
  - Less than 31 characters
  - Cannot use reserved keywords
    - auto, break, case, …, void, volatile, while

Variable Declaration in C

- **Variable**

  A name associated with a memory cell whose value can change.

- **Variable Declaration**

  Statements that communicate to the compiler the names of variables in the program and the type of information stored in each variable.

Giving a Value to a Variable

You can assign (give) a value to a variable by using the assignment operator =

VARIABLE DECLARATIONS

```
char    code ;
int     i;
long    national_debt;
float   payRate;
double  pi;
```

VALID ASSIGNMENT STATEMENTS

```
code = 'B';
i  =  14;
national_debt = 10000000;
pay_rate = 14.25;
pi = 3.1415926536;
```

Arithmetic Operators

1. ( )
   - Parenthesis (highest precedence)
2. *, /, %
   - Multiplication, division, modulus
   - Evaluated left to right
3. +, -
   - Addition and subtraction
   - Evaluated left to right

Arithmetic Operators

- Integer division produces an integer result
  - 1 / 2 evaluates to 0 (no rounding up!)
  - 19 / 5 evaluates to 3 (no rounding up!)
- Modulus (%) = remainder after division
  - 1 % 2 evaluates to 1
  - 17 % 5 evaluates to 2
- Implicit conversion – if an operation has operands of different types, the "narrower" one will be converted to the "wider" one
  - 2.0 / 5 evaluates to 0.4 (a float)

Equality, Relational Operators

- Equality operators
  - == (equal) – common error: = instead of ==
  - != (not equal)
- Relational operators
  - > (greater than)
  - < (less than)
  - >= (greater than or equal)
  - <= (less than or equal)
- Will return a 0 (false) or 1(true)

if Control Structure

- A program can make a decision using the if control structure and the equality or relational operators

```
#include <stdio.h>
int main(void)
{
    if(1 > 2) printf("%d > %d\n",1,2);
    if(1 <= 2) printf("%d <= %d\n",1,2);
    return 0;
}
```

Input in C

```
#include <stdio.h>
int main(void){
    int x = 0;
    printf("Enter an integer: ");
    scanf("%d", &x);
    printf("The integer is %d\n", x);
    return 0;
}
```

scanf Function

- `scanf("%d", &x);`
  - First argument: format control string
    - Indicates type of data to be entered
  - Second argument: location in memory
    - Use an ampersand (&) to give the address where the variable is stored
  - The computer will wait for the user to enter a value & push the enter key

printf/scanf function

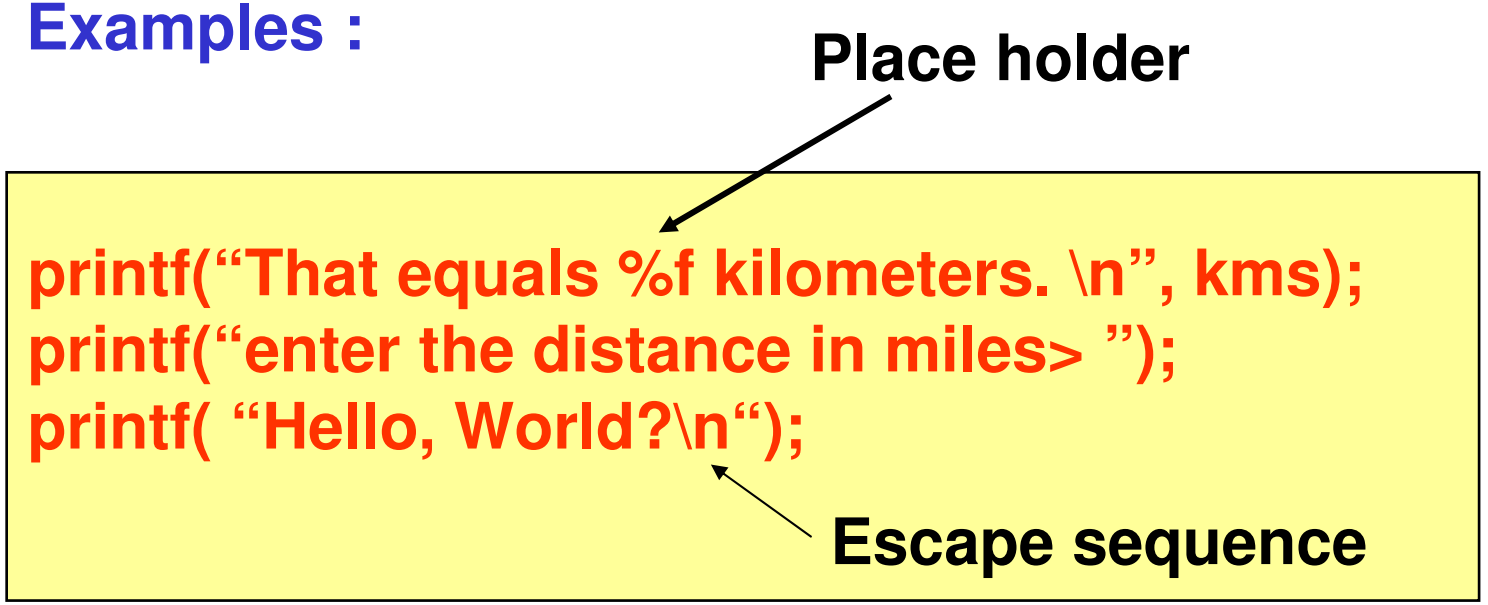
- **Printf (Output Function)**
- **The printf function displays the value of its format string after substituting in left-to-right order the values of the expressions in the print list for their list for their placeholders in the format string and after replacing escape sequences such as \n by their meanings.**
- **Scanf (Input Function)**
- **The scanf function copies into memory data entered during the program execution.**
- **The order of the placeholders must correspond to the order of the variables in the input list.**
- **The data must be entered in the same order in the input list.**
- **You should insert one or more blank characters or carriage returns between numeric items.**

Output Function

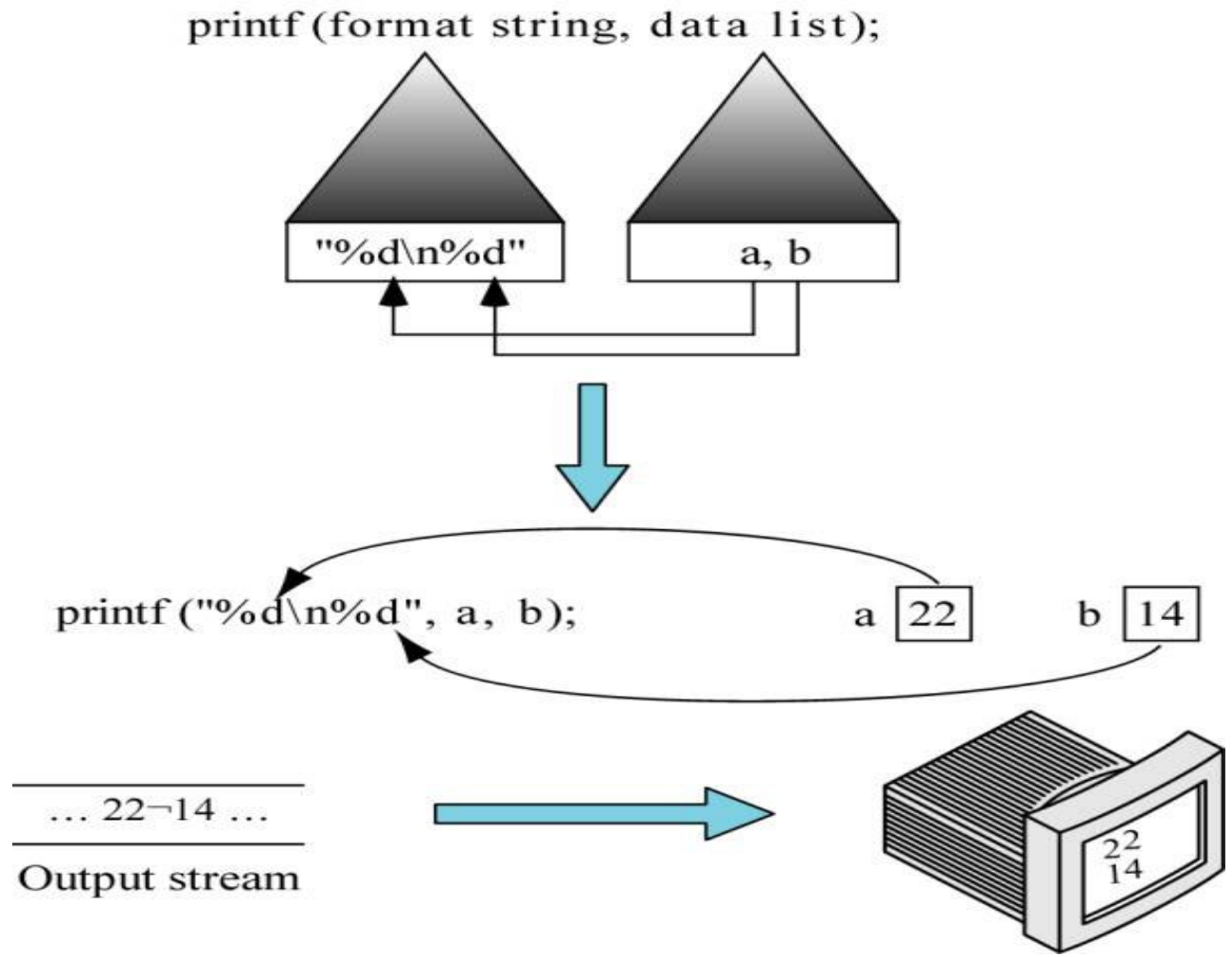
SYNTAX

```
printf( format string , print list ) ;
Printf(format string);
```

Examples :



Output Function

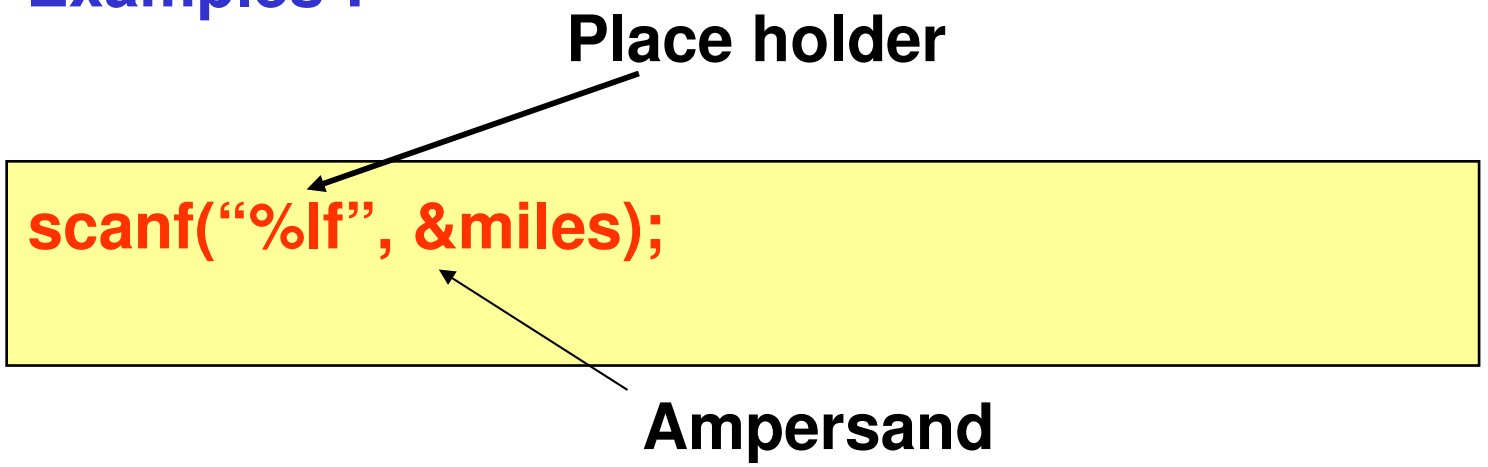


Input Function

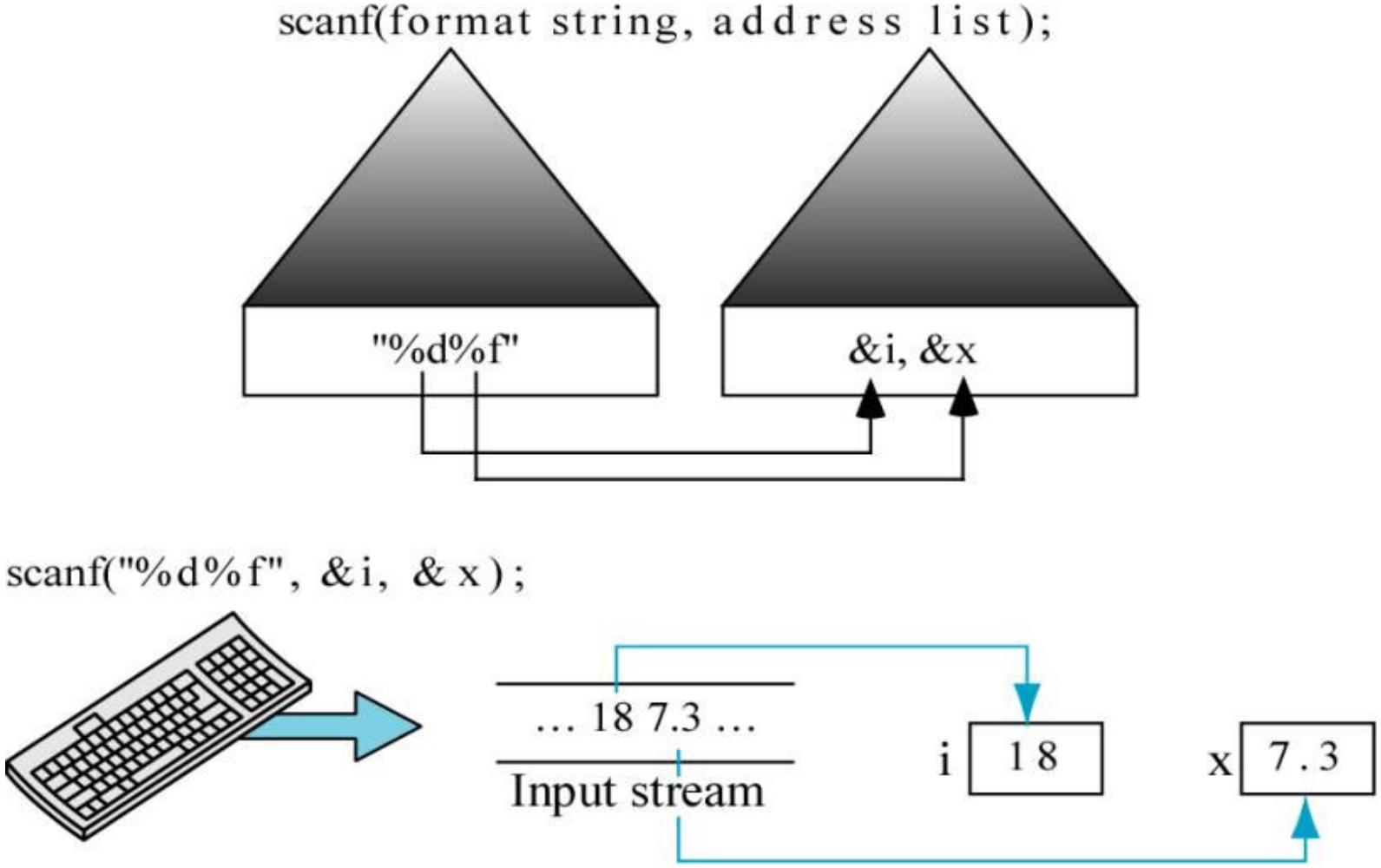
SYNTAX

```
scanf( format string , input list ) ;
```

Examples :



Input Function



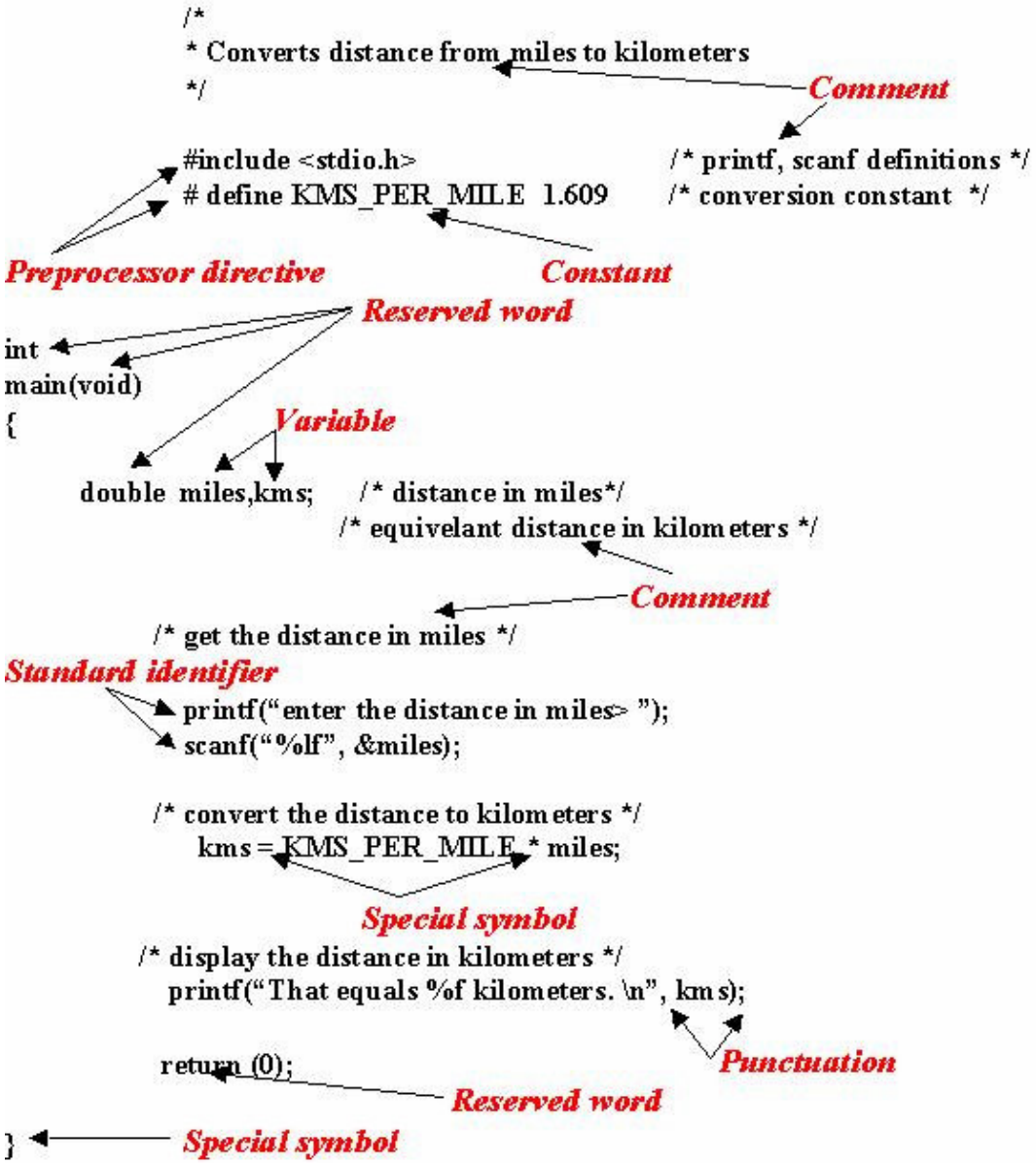
Place Holders

A placeholder is a symbol beginning with % in a format string that indicates where to display the output value

Placeholder	Variable type	Function use
%C	char	Printf/scanf
%d	int	Printf/scanf
%f	double	printf
%lf	double	scanf

Example for Another C Program

- Write a program to Convert given distance in miles to kilometers.



C Language Elements
Preprocessor Directives

Examples

```
#define KMS_PER_MILE  1.609
# define AT  '@'
# define VOTING_AGE   18
```

- **Define is a preprocessor directive.**
- **Valid constant declarations**
- **A named constant is a location in memory that we can refer to by a name, and in which a data value that cannot be changed is stored.**
- **This directives instructs the processor to replace each occurrence of KMS_PER_MILE in the text of the C program by 1.609 before compilation begins.**

Outline

- fig02_05.c

```
1  /* Fig. 2.5: fig02_05.c
2     Addition program */
3  #include <stdio.h>
4
5  /* function main begins program execution */
6  int main( void )
7  {
8     int integer1; /* first number to be input by user  */
9     int integer2; /* second number to be input by user */
10    int sum;      /* variable in which sum will be stored */
11
12    printf( "Enter first integer\n" ); /* prompt */
13    scanf( "%d", &integer1 );          /* read an integer */
14
15    printf( "Enter second integer\n" ); /* prompt */
16    scanf( "%d", &integer2 );           /* read an integer */
17
18    sum = integer1 + integer2; /* assign total to sum */
19
20    printf( "Sum is %d\n", sum ); /* print sum */
21
22    return 0;  /* indicate that program ended successfully */
23
24 } /* end function main */
```

```
Enter first integer
45
Enter second integer
72
Sum is 117
```

Definitions of variables

scanf obtains a value from the user and assigns it to **integer1**

scanf obtains a value from the user and assigns it to **integer2**

Assigns a value to **sum**

Operator(s)	Operation(s)	Order of evaluation (precedence)
( )	Parentheses	Evaluated first. If the parentheses are nested, the expression in the innermost pair is evaluated first. If there are several pairs of parentheses “on the same level” (i.e., not nested), they are evaluated left to right.
*	Multiplication	Evaluated second. If there are several, they are evaluated left to right.
/	Division	
%	Remainder	
+	Addition	Evaluated last. If there are several, they are evaluated left to right.
-	Subtraction	

Fig. 2.10 | **Precedence of arithmetic operators.**

Fig. 2.12 | Equality and relational operators.

Standard algebraic equality operator or relational operator	C equality or relational operator	Example of C condition	Meaning of C condition
Equality operators			
=	==	x == y	x is equal to y
≠	!=	x != y	x is not equal to y
Relational operators			
>	>	x > y	x is greater than y
<	<	x < y	x is less than y
≥	>=	x >= y	x is greater than or equal to y
≤	<=	x <= y	x is less than or equal to y

```c
/* Fig. 2.13: fig02_13.c
   Using if statements, relational
   operators, and equality operators */
#include <stdio.h>

/* function main begins program execution */
int main( void )
{
   int num1; /* first number to be read from user  */
   int num2; /* second number to be read from user */

   printf( "Enter two integers, and I will tell you\n" );
   printf( "the relationships they satisfy: " );

   scanf( "%d%d", &num1, &num2 ); /* read two integers */

   if ( num1 == num2 ) {
      printf( "%d is equal to %d\n", num1, num2 );
   } /* end if */

   if ( num1 != num2 ) {
      printf( "%d is not equal to %d\n", num1, num2 );
   } /* end if */

   if ( num1 < num2 ) {
      printf( "%d is less than %d\n", num1, num2 );
   } /* end if */

```

Checks if **num1** is equal to **num2**

Checks if **num1** is not equal to **num2**

Checks if **num1** is less than **num2**

```
29    if ( num1 > num2 ) {
30       printf( "%d is greater than %d\n", num1, num2 );
31    } /* end if */
32
33    if ( num1 <= num2 ) {
34       printf( "%d is less than or equal to %d\n", num1, num2 );
35    } /* end if */
36
37    if ( num1 >= num2 ) {
38       printf( "%d is greater than or equal to %d\n", num1, num2 );
39    } /* end if */
40
41    return 0;   /* indicate that program ended successfully */
42
43 } /* end function main */
43 } /* end function main */
```

Checks if **num1** is greater than **num2**

Checks if **num1** is less than or equal to **num2**

Checks if **num1** is greater than equal to **num2**

```
Enter two integers, and I will tell you
the relationships they satisfy: 3 7
3 is not equal to 7
3 is less than 7
```

(continued from previous slide…)

```
Enter two integers, and I will tell you
the relationships they satisfy:
22 is not equal to 12
22 is greater than 12
22 is greater than or equal to 12
```

```
Enter two integers, and I will tell you
the relationships they satisfy:
7 is equal to 7
7 is less than or equal to 7
7 is greater than or equal to 7
```

Conversion specifier	Description
`d`	Display as a signed decimal integer.
`i`	Display as a signed decimal integer. [*Note:* The `i` and `d` specifiers are different when used with `scanf`.]
`o`	Display as an unsigned octal integer.
`u`	Display as an unsigned decimal integer.
`x` or `X`	Display as an unsigned hexadecimal integer. `X` causes the digits `0-9` and the letters `A-F` to be displayed and `x` causes the digits `0-9` and `a-f` to be displayed.
`h` or `l` (letter `l`)	Place before any integer conversion specifier to indicate that a `short` or `long` integer is displayed, respectively. Letters `h` and `l` are more precisely called **length modifiers**.

Conversion specifier	Description
`e` or `E`	Display a floating-point value in exponential notation.
`f`	Display floating-point values in fixed-point notation.
`g` or `G`	Display a floating-point value in either the floating-point form `f` or the exponential form `e` (or `E`), based on the magnitude of the value.
`L`	Place before any floating-point conversion specifier to indicate that a `long double` floating-point value is displayed.

Printing Strings and Characters

- `c`
  - Prints `char` argument
  - Cannot be used to print the first character of a string
- `s`
  - Requires a pointer to `char` as an argument
  - Prints characters until `NULL` (`'\0'`) encountered
  - Cannot print a `char` argument
- Remember
  - Single quotes for character constants (`'z'`)
  - Double quotes for strings `"z"` (which actually contains two characters, `'z'` and `'\0'`)
- `p`
  - Displays pointer value (address)
- `n`
  - Stores number of characters already output by current `printf` statement
  - Takes a pointer to an integer as an argument
  - Nothing printed by a `%n` specification
  - Every `printf` call returns a value
    - Number of characters output
    - Negative number if error occurs
- `%`
  - Prints a percent sign
  - %%

```c
#include <stdio.h>

int main( void )
{
   int *ptr;      /* define pointer to int */
   int x = 12345; /* initialize int x */
   int y;         /* define int y */

   ptr = &x;      /* assign address of x to ptr */
   printf( "The value of ptr is %p\n", ptr );
   printf( "The address of x is %p\n\n", &x );

   printf( "Total characters printed on this line:%n", &y );
   printf( " %d\n\n", y );

   y = printf( "This line has 28 characters\n%n" );
   printf( "%d characters were printed\n\n", y );

   printf( "Printing a %% in a format control string\n" );

   return 0; /* indicates successful termination */

} /* end main */
```

p specifies a memory address will be printed

n stores the number of characters printed on a line

% prints a percent sign

```
The value of ptr is 0012FF78
The address of x is 0012FF78

Total characters printed on this line: 38

This line has 28 characters
28 characters were printed

Printing a % in a format control string
```

9.8 Printing with Field Widths and Precision

- Field width
  - Size of field in which data is printed
  - If width larger than data, default right justified
    - If field width too small, increases to fit data
    - Minus sign uses one character position in field
  - Integer width inserted between % and conversion specifier
  - `%4d` – field width of 4

```c
/* Fig 9.8: fig09_08.c */
/* Printing integers right-justified */
#include <stdio.h>

int main( void )
{
   printf( "%4d\n", 1 );
   printf( "%4d\n", 12 );
   printf( "%4d\n", 123 );
   printf( "%4d\n", 1234 );
   printf( "%4d\n\n", 12345 );

   printf( "%4d\n", -1 );
   printf( "%4d\n", -12 );
   printf( "%4d\n", -123 );
   printf( "%4d\n", -1234 );
   printf( "%4d\n", -12345 );

   return 0; /* indicates successful termination */

} /* end main */
```

A field width of 4 will make C attempt to print the number in a 4-character space

Note that C considers the minus sign a character

```
   1
  12
 123
1234
12345

  -1
 -12
-123
-1234
-12345
```

9.8 Printing with Field Widths and Precision

- Precision
  - Meaning varies depending on data type
  - Integers (default `1`)
    - Minimum number of digits to print
      - If data too small, prefixed with zeros
  - Floating point
    - Number of digits to appear after decimal (`e` and `f`)
      - For `g` – maximum number of significant digits
  - Strings
    - Maximum number of characters to be written from string
  - Format
    - Use a dot (`.`) then precision number after `%`

```
%.3f
```

```c
/* Fig 9.9: fig09_09.c */
/* Using precision while printing integers,
   floating-point numbers, and strings */
#include <stdio.h>

int main( void )
{
   int i = 873;                  /* initialize int i */
   double f = 123.94536;         /* initialize double f */
   char s[] = "Happy Birthday"; /* initialize char array s */

   printf( "Using precision for integers\n" );
   printf( "\t%.4d\n\t%.9d\n\n", i, i );

   printf( "Using precision for floating-point numbers\n" );
   printf( "\t%.3f\n\t%.3e\n\t%.3g\n\n", f, f, f );
```

Precision for integers specifies the minimum number of characters to be printed

Precision for the **g** specifier controls the maximum number of significant digits printed

Precision for **f** and **e** specifiers controls the number of digits after the decimal point

```
17
18    printf( "Using precision for strings\n" );
19    printf( "\t%.11s\n", s );
20
21    return 0; /* indicates successful termination */
22
23 } /* end main */
```

Precision for strings specifies the maximum number of characters to be printed

```
Using precision for integers
        0873
        000000873

Using precision for floating-point numbers
        123.945
        1.239e+002
        124

Using precision for strings
        Happy Birth
```

Escape sequence	Description
`\'` (single quote)	Output the single quote (') character.
`\"` (double quote)	Output the double quote (") character.
`\?` (question mark)	Output the question mark (?) character.
`\\` (backslash)	Output the backslash (\) character.
`\a` (alert or bell)	Cause an audible (bell) or visual alert.
`\b` (backspace)	Move the cursor back one position on the current line.
`\f` (new page or form feed)	Move the cursor to the start of the next logical page.
`\n` (newline)	Move the cursor to the beginning of the next line.
`\r` (carriage return)	Move the cursor to the beginning of the current line.
`\t` (horizontal tab)	Move the cursor to the next horizontal tab position.
`\v` (vertical tab)	Move the cursor to the next vertical tab position.

Fig. 9.16 | **Escape sequences.**

```c
/* Fig 9.18: fig09_18.c */
/* Reading integers */
#include <stdio.h>

int main( void )
{
   int a;
   int b;
   int c;
   int d;
   int e;
   int f;
   int g;

   printf( "Enter seven integers: " );
   scanf( "%d%i%i%i%o%u%x", &a, &b, &c, &d, &e, &f, &g );

   printf( "The input displayed as decimal integers is:\n" );
   printf( "%d %d %d %d %d %d %d\n", a, b, c, d, e, f, g );

   return 0; /* indicates successful termination */

} /* end main */
```

d specifies a decimal integer will be input

i specifies an integer will be input

o specifies an octal integer will be input

u specifies an unsigned decimal integer will be input

x specifies a hexadecimal integer will be input

```
Enter seven integers: -70 -70 070 0x70 70 70 70
The input displayed as decimal integers is:
-70 -70 56 112 56 70 112
```

```c
/* Fig 9.19: fig09_19.c */
/* Reading floating-point numbers */
#include <stdio.h>

/* function main begins program execution */
int main( void )
{
   double a;
   double b;
   double c;

   printf( "Enter three floating-point numbers: \n" );
   scanf( "%le%lf%lg", &a, &b, &c );

   printf( "Here are the numbers entered in plain\n" );
   printf( "floating-point notation:\n" );
   printf( "%f\n%f\n%f\n", a, b, c );

   return 0; /* indicates successful termination */

} /* end main */
```

e, **f**, and **g** specify a floating-point number will be input

l specifies a **double** or **long double** will be input

```
Enter three floating-point numbers:
1.27987 1.27987e+03 3.38476e-06
Here are the numbers entered in plain
floating-point notation:
1.279870
1279.870000
0.000003
```

```c
/* Fig 9.20: fig09_20.c */
/* Reading characters and strings */
#include <stdio.h>

int main( void )
{
   char x;
   char y[ 9 ];

   printf( "Enter a string: " );
   scanf( "%c%s", &x, y );

   printf( "The input was:\n" );
   printf( "the character \"%c\" ", x );
   printf( "and the string \"%s\"\n", y );

   return 0; /* indicates successful termination */

} /* end main */
```

c specifies a character will be input

s specifies a string will be input

```
Enter a string: Sunday
The input was:
the character "S" and the string "unday"
```

```c
/* Fig 9.21: fig09_21.c */
/* Using a scan set */
#include <stdio.h>

/* function main begins program execution */
int main( void )
{
   char z[ 9 ]; /* define array z */

   printf( "Enter string: " );
   scanf( "%[aeiou]", z ); /* search for set of characters */

   printf( "The input was \"%s\"\n", z );

   return 0; /* indicates successful termination */

} /* end main */
```

```
Enter string: ooeeooahah
The input was "ooeeooa"
```

`[]` specifies only the initial segment of a string that contains the characters in brackets will be read

```c
/* Fig 9.22: fig09_22.c */
/* Using an inverted scan set */
#include <stdio.h>

int main( void )
{
   char z[ 9 ];

   printf( "Enter a string: " );
   scanf( "%[^aeiou]", z ); /* inverted scan set */

   printf( "The input was \"%s\"\n", z );

   return 0; /* indicates successful termination */

} /* end main */
```

```
Enter a string: String
The input was "Str"
```

[] and **^** specify only the initial segment of a string that does **not** contain the characters in brackets will be read

```c
/* Fig 9.23: fig09_23.c */
/* inputting data with a field width */
#include <stdio.h>

int main( void )
{
   int x;
   int y;

   printf( "Enter a six digit integer: " );
   scanf( "%2d%d", &x, &y );

   printf( "The integers input were %d and %d\n", x, y );

   return 0; /* indicates successful termination */

} /* end main */
```

A field width of 2 tells C to only read the first 2 characters of that input

```
Enter a six digit integer: 123456
The integers input were 12 and 3456
```

```c
/* Fig 9.24: fig09_24.c */
/* Reading and discarding characters from the input stream */
#include <stdio.h>

int main( void )
{
   int month1;
   int day1;
   int year1;
   int month2;
   int day2;
   int year2;

   printf( "Enter a date in the form mm-dd-yyyy: " );
   scanf( "%d%*c%d%*c%d", &month1, &day1, &year1 );

   printf( "month = %d  day = %d  year = %d\n\n", month1, day1, year1 );

   printf( "Enter a date in the form mm/dd/yyyy: " );
   scanf( "%d%*c%d%*c%d", &month2, &day2, &year2 );

   printf( "month = %d  day = %d  year = %d\n", month2, day2, year2 );

   return 0; /* indicates successful termination */

} /* end main */
```

***** is a wildcard—**scanf** will disregard anything between the two inputs on either side of it

```
Enter a date in the form mm-dd-yyyy: 11-18-2003
month = 11  day = 18  year = 2003

Enter a date in the form mm/dd/yyyy: 11/18/2003
month = 11  day = 18  year = 2003
```