CS112 - *Software Engineering Division*
Programming Languages I

Lecture 5:
Introduction to Pointers (Part II)

Amr S. Ghoneim
(Assistant Professor, Computer Science Dept.)

Helwan University
Spring “Semester 2” 2017-18

Lecture is based on its counterparts in the following courses/resources:

○ *C How to Program - Lecture Notes*, by **Deitel & Associates, Inc. and Pearson Education Inc.**

○ *Introduction to Programming & Problem Solving*, **British University in Egypt** (School of Informatics & Computer Sciences)

Pointer Initialization

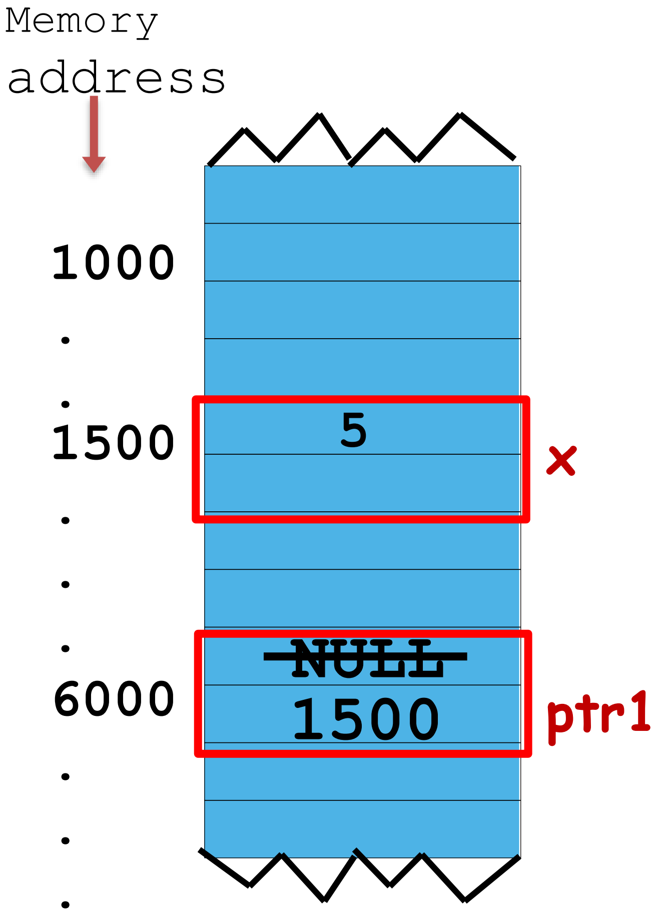
NULL

-Means points to ***nothing*** -Defined in <stdio.h>

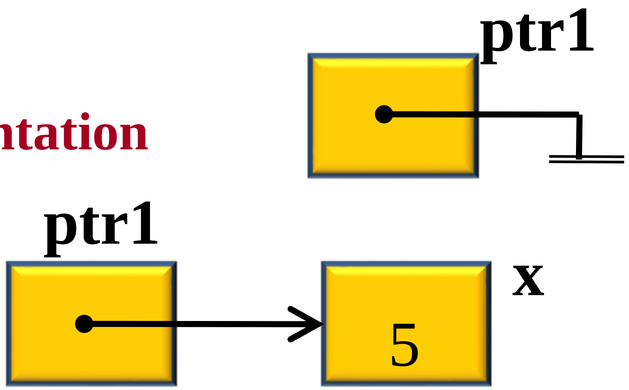
```
int x =5;
int *ptr1=NULL;
// pointer initialization
ptr1= &x;
printf (“address of x is %p\n", &x );
printf (“ptr1 value is %p", ptr1 );
```

For address

For address



Logical representation



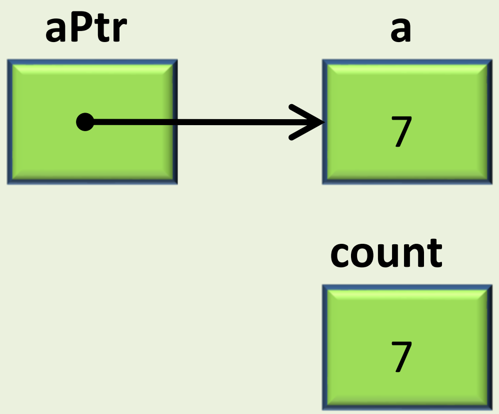
Address of x is 1500
ptr1 **value** is 1500

Pointers operators: & and *

Indirection (dereferencing) operator

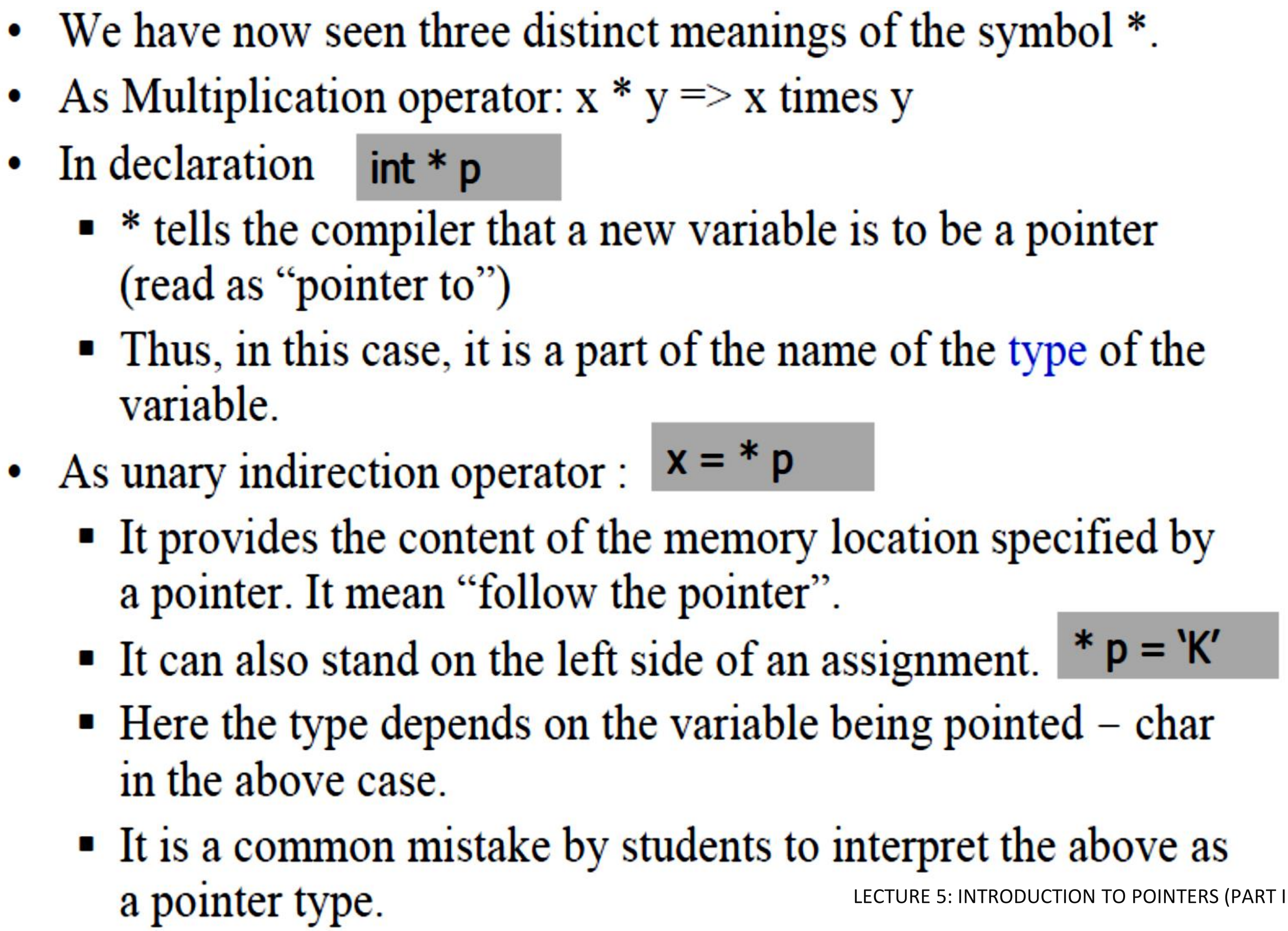
```
int count, a;
int *aPtr;

a = 7;
aPtr = &a;
count=*aPtr;

printf( "\nThe value of count is %d", count);
```

The value of variable it points to



```
The value of count is 7
```

Pointers operators: 3 Uses of the operator *



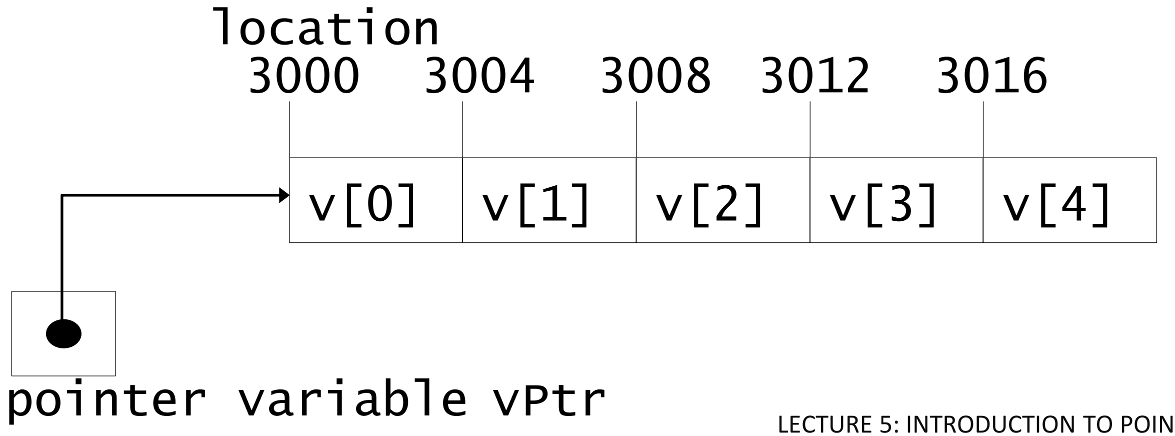
- We have now seen three distinct meanings of the symbol *.
- As Multiplication operator: x * y => x times y
- In declaration `int * p`
  - * tells the compiler that a new variable is to be a pointer (read as “pointer to”)
  - Thus, in this case, it is a part of the name of the type of the variable.
- As unary indirection operator : `x = * p`
  - It provides the content of the memory location specified by a pointer. It mean “follow the pointer”.
  - It can also stand on the left side of an assignment. `* p = ‘K’`
  - Here the type depends on the variable being pointed – char in the above case.
  - It is a common mistake by students to interpret the above as a pointer type.

Pointer Expressions and Pointer Arithmetic

- Arithmetic operations can be performed on pointers
  - Increment/decrement pointer (++ or −−)
  - Add an integer to a pointer(+ or += , − or −=)
  - Pointers may be subtracted from each other
  - Operations meaningless unless performed on an array

Pointer Expressions and Pointer Arithmetic

- 5 element `int` array on machine with 4 byte `ints`
  - `vPtr` points to first element `v[ 0 ]`
    - at location `3000 (vPtr = 3000)`
  - `vPtr += 2;` sets `vPtr` to `3008`
    - `vPtr` points to `v[ 2 ]` (incremented by 2), but the machine has 4 byte `ints`, so it points to address `3008`



location
3000 3004 3008 3012 3016

v[0] v[1] v[2] v[3] v[4]

pointer variable vPtr

Pointer Expressions and Pointer Arithmetic

- Subtracting pointers
  - Returns number of elements from one to the other. If
    - `vPtr2 = v[ 2 ];`
    - `vPtr = v[ 0 ];`
  - `vPtr2 - vPtr` would produce 2

Pointer Expressions and Pointer Arithmetic

- Pointers of the same type can be assigned to each other
  - If not the same type, a cast operator must be used
  - Exception: pointer to `void` (type `void *`)
    - Generic pointer, represents any type
    - No casting needed to convert a pointer to `void` pointer
    - `void` pointers cannot be dereferenced

The Relationship Between Pointers and Arrays

- Arrays and pointers are closely related
  - Array name is just like a constant pointer
  - Pointers can do array subscripting operations
- Define an array `b[ 5 ]` and a pointer `bPtr`
  - To set them equal to one another use:
      - `bPtr = b;`
    - The array name (b) is actually the address of first element of the array `b[ 5 ]`
      - `bPtr = &b[ 0 ]`
    - Explicitly assigns `bPtr` to address of first element of b

The Relationship Between Pointers and Arrays

- Element `b[ 3 ]`
  - Can be accessed by `*( bPtr + 3 )`
    - Where `n` is the offset. Called pointer/offset notation
  - Can be accessed by `bptr[ 3 ]`
    - Called pointer/subscript notation
    - `bPtr[ 3 ]` same as `b[ 3 ]`
  - Can be accessed by performing pointer arithmetic on the array itself
    - `*( b + 3 )`

```
/* Fig. 7.20: fig07_20.cpp
   Using subscripting and pointer notations with arrays */

#include <stdio.h>

int main()
{
   int b[] = { 10, 20, 30, 40 }; /* initialize array b */
   int *bPtr = b;                /* set bPtr to point to array b */
   int i;                        /* counter */
   int offset;                   /* counter */

   /* output array b using array subscript notation */
   printf( "Array b printed with:\nArray subscript notation\n" );

   /* loop through array b */
   for ( i = 0; i < 4; i++ ) {
      printf( "b[ %d ] = %d\n", i, b[ i ] );
   } /* end for */

   /* output array b using array name and pointer/offset notation */
   printf( "\nPointer/offset notation where\n"
           "the pointer is the array name\n" );

```

```c
   /* loop through array b */
   for ( offset = 0; offset < 4; offset++ ) {
      printf( "*( b + %d ) = %d\n", offset, *( b + offset ) );
   } /* end for */

   /* output array b using bPtr and array subscript notation */
   printf( "\nPointer subscript notation\n" );

   /* loop through array b */
   for ( i = 0; i < 4; i++ ) {
      printf( "bPtr[ %d ] = %d\n", i, bPtr[ i ] );
   } /* end for */

   /* output array b using bPtr and pointer/offset notation */
   printf( "\nPointer/offset notation\n" );

   /* loop through array b */
   for ( offset = 0; offset < 4; offset++ ) {
      printf( "*( bPtr + %d ) = %d\n", offset, *( bPtr + offset ) );
   } /* end for */

   return 0; /* indicates successful termination */

} /* end main */
```

Program Output:

```
Array b printed with:
Array subscript notation
b[ 0 ] = 10
b[ 1 ] = 20
b[ 2 ] = 30
b[ 3 ] = 40

Pointer/offset notation where
the pointer is the array name
*( b + 0 ) = 10
*( b + 1 ) = 20
*( b + 2 ) = 30
*( b + 3 ) = 40

Pointer subscript notation
bPtr[ 0 ] = 10
bPtr[ 1 ] = 20
bPtr[ 2 ] = 30
bPtr[ 3 ] = 40

Pointer/offset notation
*( bPtr + 0 ) = 10
*( bPtr + 1 ) = 20
*( bPtr + 2 ) = 30
*( bPtr + 3 ) = 40
```

```c
/* Fig. 7.21: fig07_21.c
   Copying a string using array notation and pointer notation. */
#include <stdio.h>

void copy1( char *s1, const char *s2 ); /* prototype */
void copy2( char *s1, const char *s2 ); /* prototype */

int main()
{
   char string1[ 10 ];          /* create array string1 */
   char *string2 = "Hello";     /* create a pointer to a string */
   char string3[ 10 ];          /* create array string3 */
   char string4[] = "Good Bye"; /* create a pointer to a string */

   copy1( string1, string2 );
   printf( "string1 = %s\n", string1 );

   copy2( string3, string4 );
   printf( "string3 = %s\n", string3 );

   return 0; /* indicates successful termination */

} /* end main */

```

```c
/* copy s2 to s1 using array notation */
void copy1( char *s1, const char *s2 )
{
   int i; /* counter */

   /* loop through strings */
   for ( i = 0; ( s1[ i ] = s2[ i ] ) != '\0'; i++ ) {
      ; /* do nothing in body */
   } /* end for */

} /* end function copy1 */

/* copy s2 to s1 using pointer notation */
void copy2( char *s1, const char *s2 )
{
   /* loop through strings */
   for ( ; ( *s1 = *s2 ) != '\0'; s1++, s2++ ) {
      ; /* do nothing in body */
   } /* end for */

} /* end function copy2 */
```

Program Output:

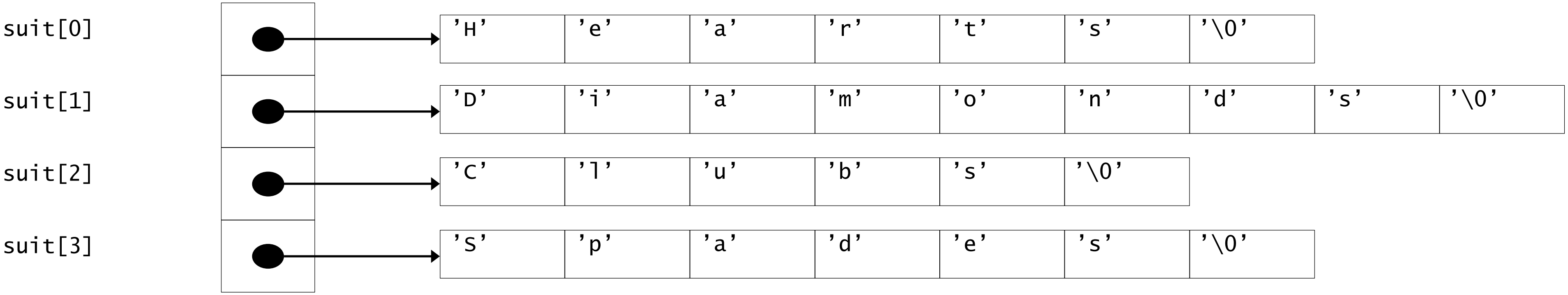
```
string1 = Hello
string3 = Good Bye
```

Arrays of Pointers

- Arrays can contain pointers
- For example: an array of strings
  - `char *suit[ 4 ] = { "Hearts", "Diamonds", "Clubs", "Spades" };`
  - Strings are pointers to the first character
  - `char *` – each element of `suit` is a pointer to a `char`
  - The strings are not actually stored in the array `suit`, only pointers to the strings are stored

Arrays of Pointers

○ `suit` array has a fixed size, but strings can be of any size



suit[0]	'H'	'e'	'a'	'r'	't'	's'	'\0'		
suit[1]	'D'	'i'	'a'	'm'	'o'	'n'	'd'	's'	'\0'
suit[2]	'C'	'l'	'u'	'b'	's'	'\0'			
suit[3]	'S'	'p'	'a'	'd'	'e'	's'	'\0'		

Thank You!

Questions?

CS112 - *Software Engineering Division*
Programming Languages I

Lecture 6:
Characters and Strings (Part I)

Amr S. Ghoneim
(Assistant Professor, Computer Science Dept.)

Helwan University
Spring "Semester 2" 2017-18

Lecture is based on its counterparts in the following courses/resources:
- *C How to Program - Lecture Notes*, by **Deitel & Associates, Inc. and Pearson Education Inc.**
- *Introduction to Programming & Problem Solving*, **British University in Egypt** (School of Informatics & Computer Sciences)

Characters and Strings

- Outline
- Introduction
- Fundamentals of Strings and Characters
- Character Handling Library
- String Conversion Functions
- Standard Input / Output Library Functions
- String Manipulation Functions of the String Handling Library
- Comparison Functions of the String Handling Library
- Search Functions of the String Handling Library
- Memory Functions of the String Handling Library
- Other Functions of the String Handling Library

Objectives

- In this chapter, you will learn:
  - To be able to use the functions of the character handling library (ctype).
  - To be able to use the string and character input/output functions of the standard input/output library (stdio).
  - To be able to use the string conversion functions of the general utilities library (stdlib).
  - To be able to use the string processing functions of the string handling library (string).
  - To appreciate the power of function libraries as a means of achieving software reusability.

Introduction

- Introduce some standard library functions
  - Easy string and character processing
  - Programs can process characters, strings, lines of text, and blocks of memory
- These techniques used to make
  - For example: Word processors

Fundamentals of Strings and Characters

- Characters
  - Building blocks of programs
    - Every program is a sequence of meaningfully grouped characters
  - Character constant
    - An `int` value represented as a character in single quotes
    - `'z'` represents the integer value of `z`
- Strings
  - Series of characters treated as a single unit
    - Can include letters, digits and special characters `(*,/,$)`
  - String literal (string constant) - written in double quotes
    - `"Hello"`
  - Strings are arrays of characters
    - String a pointer to first character

Fundamentals of Strings and Characters

- String definitions
  - Define as a character array or a variable of type `char *`
    - `char color[] = "blue";`
    - `char *colorPtr = "blue";`
  - Remember that strings represented as character arrays end with `'\0'`
    - `color` has 5 elements
- Inputting strings
  - Use `scanf`
    - `scanf("%s", word);`
    - Copies input into `word[]`
    - Do not need & (because a string is a pointer)
  - Remember to leave room in the array for `'\0'`

Character Handling Library

- Character handling library
  - Includes functions to perform useful tests and manipulations of character data
  - Each function receives a character (an `int`) or EOF as an argument
- The following slide contains a table of all the functions in `<ctype.h>`

Character Handling Library

Prototype	Description
`int isdigit( int c );`	Returns `true` if `c` is a digit and `false` otherwise.
`int isalpha( int c );`	Returns `true` if `c` is a letter and `false` otherwise.
`int isalnum( int c );`	Returns `true` if `c` is a digit or a letter and `false` otherwise.
`int isxdigit( int c );`	Returns `true` if `c` is a hexadecimal digit character and `false` otherwise.
`int islower( int c );`	Returns `true` if `c` is a lowercase letter and `false` otherwise.
`int isupper( int c );`	Returns `true` if `c` is an uppercase letter; `false` otherwise.
`int tolower( int c );`	If `c` is an uppercase letter, `tolower` returns `c` as a lowercase letter. Otherwise, `tolower` returns the argument unchanged.
`int toupper( int c );`	If `c` is a lowercase letter, `toupper` returns `c` as an uppercase letter. Otherwise, `toupper` returns the argument unchanged.
`int isspace( int c );`	Returns `true` if `c` is a white-space character—newline (`'\n'`), space (`' '`), form feed (`'\f'`), carriage return (`'\r'`), horizontal tab (`'\t'`), or vertical tab (`'\v'`)—and `false` otherwise
`int iscntrl( int c );`	Returns `true` if `c` is a control character and `false` otherwise.
`int ispunct( int c );`	Returns `true` if `c` is a printing character other than a space, a digit, or a letter and `false` otherwise.
`int isprint( int c );`	Returns `true` value if `c` is a printing character including space (`' '`) and `false` otherwise.
`int isgraph( int c );`	Returns `true` if `c` is a printing character other than space (`' '`) and `false` otherwise.

Characters & Strings Example 1

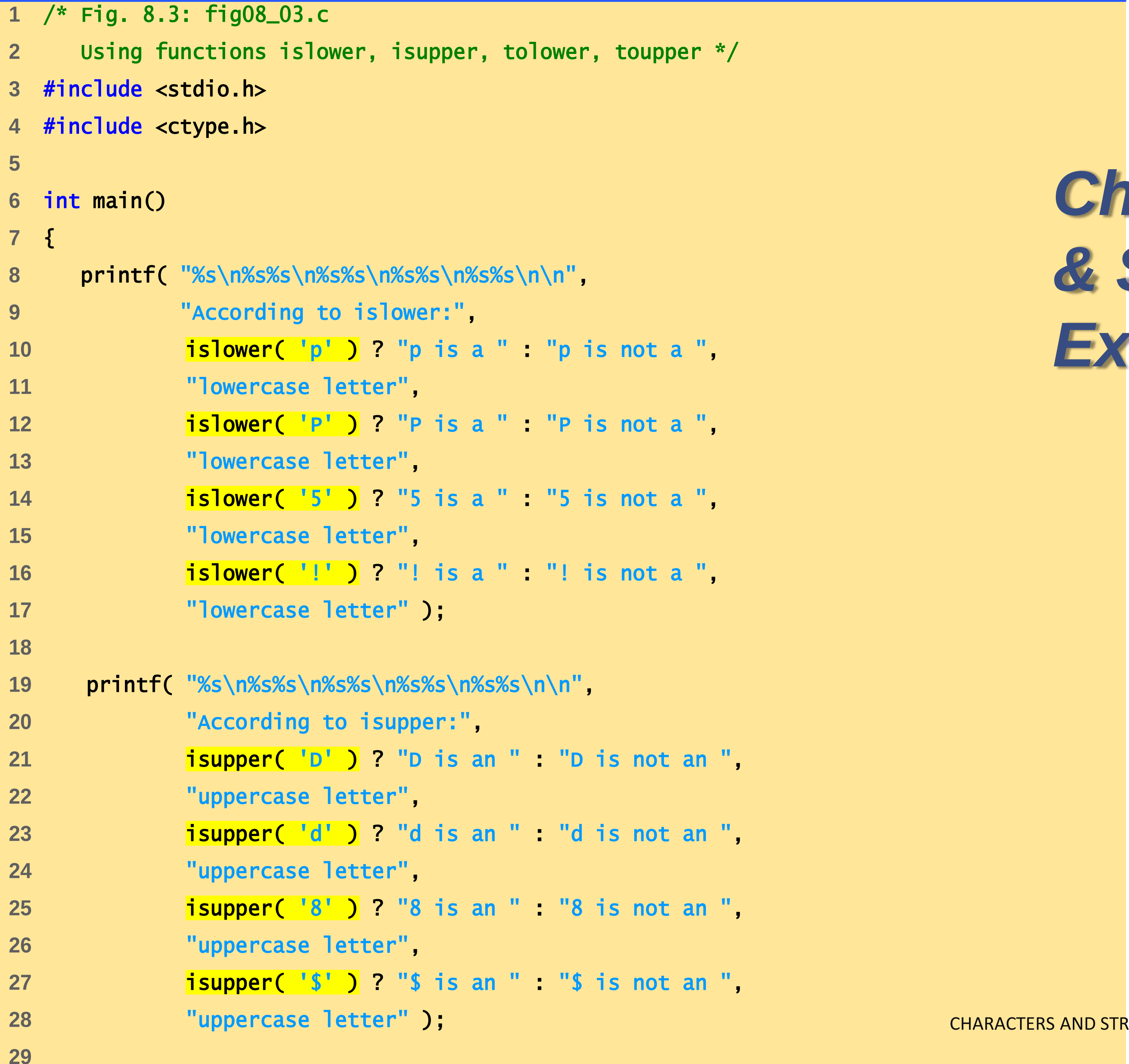
```c
/* Fig. 8.2: fig08_02.c
   Using functions isdigit, isalpha, isalnum, and isxdigit */
#include <stdio.h>
#include <ctype.h>

int main()
{
   printf( "%s\n%s%s\n%s%s\n\n", "According to isdigit: ",
       isdigit( '8' ) ? "8 is a " : "8 is not a ", "digit",
        isdigit( '#' ) ? "# is a " : "# is not a ", "digit" );

    printf( "%s\n%s%s\n%s%s\n%s%s\n%s%s\n\n",
        "According to isalpha:",
        isalpha( 'A' ) ? "A is a " : "A is not a ", "letter",
        isalpha( 'b' ) ? "b is a " : "b is not a ", "letter",
        isalpha( '&' ) ? "& is a " : "& is not a ", "letter",
        isalpha( '4' ) ? "4 is a " : "4 is not a ", "letter" );

    printf( "%s\n%s%s\n%s%s\n%s%s\n\n",
        "According to isalnum:",
        isalnum( 'A' ) ? "A is a " : "A is not a ",
        "digit or a letter",
        isalnum( '8' ) ? "8 is a " : "8 is not a ",
        "digit or a letter",
        isalnum( '#' ) ? "# is a " : "# is not a ",
        "digit or a letter" );

```

Characters & Strings Example 1

```c
   printf( "%s\n%s%s\n%s%s\n%s%s\n%s%s\n%s%s\n",
       "According to isxdigit:",
       isxdigit( 'F' ) ? "F is a " : "F is not a ",
       "hexadecimal digit",
       isxdigit( 'J' ) ? "J is a " : "J is not a ",
       "hexadecimal digit",
       isxdigit( '7' ) ? "7 is a " : "7 is not a ",
       "hexadecimal digit",
       isxdigit( '$' ) ? "$ is a " : "$ is not a ",
       "hexadecimal digit",
       isxdigit( 'f' ) ? "f is a " : "f is not a ",
       "hexadecimal digit" );

   return 0; /* indicates successful termination */

} /* end main */
```

Program Output:

```
According to isdigit:
8 is a digit
# is not a digit

According to isalpha:
A is a letter
b is a letter
& is not a letter
4 is not a letter

According to isalnum:
A is a digit or a letter
8 is a digit or a letter
# is not a digit or a letter

According to isxdigit:
F is a hexadecimal digit
J is not a hexadecimal digit
7 is a hexadecimal digit
$ is not a hexadecimal digit
f is a hexadecimal digit
```



```c
/* Fig. 8.3: fig08_03.c
   Using functions islower, isupper, tolower, toupper */
#include <stdio.h>
#include <ctype.h>

int main()
{
   printf( "%s\n%s%s\n%s%s\n%s%s\n%s%s\n\n",
           "According to islower:",
            islower( 'p' ) ? "p is a " : "p is not a ",
            "lowercase letter",
            islower( 'P' ) ? "P is a " : "P is not a ",
            "lowercase letter",
            islower( '5' ) ? "5 is a " : "5 is not a ",
            "lowercase letter",
            islower( '!' ) ? "! is a " : "! is not a ",
            "lowercase letter" );

    printf( "%s\n%s%s\n%s%s\n%s%s\n%s%s\n\n",
            "According to isupper:",
            isupper( 'D' ) ? "D is an " : "D is not an ",
            "uppercase letter",
            isupper( 'd' ) ? "d is an " : "d is not an ",
            "uppercase letter",
            isupper( '8' ) ? "8 is an " : "8 is not an ",
            "uppercase letter",
            isupper( '$' ) ? "$ is an " : "$ is not an ",
            "uppercase letter" );

```

Characters & Strings Example 2

```c
   printf( "%s%c\n%s%c\n%s%c\n%s%c\n",
           "u converted to uppercase is ", toupper( 'u' ),
           "7 converted to uppercase is ", toupper( '7' ),
           "$ converted to uppercase is ", toupper( '$' ),
           "L converted to lowercase is ", tolower( 'L' ) );

   return 0; /* indicates successful termination */

} /* end main */
```

Characters & Strings Example 2

Program Output:

```
According to islower:
p is a lowercase letter
P is not a lowercase letter
5 is not a lowercase letter
! is not a lowercase letter

According to isupper:
D is an uppercase letter
d is not an uppercase letter
8 is not an uppercase letter
$ is not an uppercase letter

u converted to uppercase is U
7 converted to uppercase is 7
$ converted to uppercase is $
L converted to lowercase is l
```

Characters & Strings Example 3

```c
/* Fig. 8.4: fig08_04.c
  Using functions isspace, iscntrl, ispunct, isprint, isgraph */
#include <stdio.h>
#include <ctype.h>

int main()
{
   printf( "%s\n%s%s%s\n%s%s%s\n%s%s\n\n",
       "According to isspace:",
        "Newline", isspace( '\n' ) ? " is a " : " is not a ",
        "whitespace character", "Horizontal tab",
        isspace( '\t' ) ? " is a " : " is not a ",
        "whitespace character",
        isspace( '%' ) ? "% is a " : "% is not a ",
        "whitespace character" );

    printf( "%s\n%s%s%s\n%s%s\n\n", "According to iscntrl:",
        "Newline", iscntrl( '\n' ) ? " is a " : " is not a ",
        "control character", iscntrl( '$' ) ? "$ is a " :
        "$ is not a ", "control character" );

```

Characters & Strings Example 3

```
   printf( "%s\n%s%s\n%s%s\n%s%s\n\n",
       "According to ispunct:",
       ispunct( ';' ) ? "; is a " : "; is not a ",
       "punctuation character",
       ispunct( 'Y' ) ? "Y is a " : "Y is not a ",
       "punctuation character",
       ispunct( '#' ) ? "# is a " : "# is not a ",
       "punctuation character" );

   printf( "%s\n%s%s\n%s%s%s\n\n", "According to isprint:",
       isprint( '$' ) ? "$ is a " : "$ is not a ",
       "printing character",
       "Alert", isprint( '\a' ) ? " is a " : " is not a ",
       "printing character" );

   printf( "%s\n%s%s\n%s%s%s\n",  "According to isgraph:",
       isgraph( 'Q' ) ? "Q is a " : "Q is not a ",
       "printing character other than a space",
       "Space", isgraph( ' ' ) ? " is a " : " is not a ",
       "printing character other than a space" );

   return 0; /* indicates successful termination */

} /* end main */
```

Program Output:

```
According to isspace:
Newline is a whitespace character
Horizontal tab is a whitespace character
% is not a whitespace character

According to iscntrl:
Newline is a control character
$ is not a control character

According to ispunct:
; is a punctuation character
Y is not a punctuation character
# is a punctuation character

According to isprint:
$ is a printing character
Alert is not a printing character

According to isgraph:
Q is a printing character other than a space
Space is not a printing character other than a space
```

String Conversion Functions

- Conversion functions
  - In `<stdlib.h>` (general utilities library)
- Convert strings of digits to integer and floating-point values

Function prototype	Function description
`double atof( const char *nPtr );`	Converts the string `nPtr` to `double`.
`int atoi( const char *nPtr );`	Converts the string `nPtr` to `int`.
`long atol( const char *nPtr );`	Converts the string `nPtr` to `long int`.

```c
/* Fig. 8.6: fig08_06.c
   Using atof */
#include <stdio.h>
#include <stdlib.h>

int main()
{
   double d; /* variable to hold converted string */

   d = atof( "99.0" );

   printf( "%s%.3f\n%s%.3f\n",
           "The string \"99.0\" converted to double is ", d,
           "The converted value divided by 2 is ",
           d / 2.0 );

   return 0; /* indicates successful termination */

} /* end main */
```

Characters & Strings Example 4

Program Output:

```
The string "99.0" converted to double is 99.000
The converted value divided by 2 is 49.500
```

Characters & Strings Example 5

```c
/* Fig. 8.7: fig08_07.c
   Using atoi */
#include <stdio.h>
#include <stdlib.h>

int main()
{
   int i; /* variable to hold converted string */

   i = atoi( "2593" );

   printf( "%s%d\n%s%d\n",
           "The string \"2593\" converted to int is ", i,
           "The converted value minus 593 is ", i - 593 );

   return 0; /* indicates successful termination */

} /* end main */
```

Program Output:

```
The string "2593" converted to int is 2593
The converted value minus 593 is 2000
```

Characters & Strings Example 6

```c
/* Fig. 8.8: fig08_08.c
   Using atol */
#include <stdio.h>
#include <stdlib.h>

int main()
{
   long l; /* variable to hold converted string */

   l = atol( "1000000" );

   printf( "%s%ld\n%s%ld\n",
           "The string \"1000000\" converted to long int is ", l,
           "The converted value divided by 2 is ", l / 2 );

   return 0; /* indicates successful termination */

} /* end main */
```

Program Output:

```
The string "1000000" converted to long int is 1000000
The converted value divided by 2 is 500000
```

Standard Input/Output Library Functions

- Functions in `<stdio.h>`
- Used to manipulate character and string data

Function prototype	Function description
`int getchar( void );`	Inputs the next character from the standard input and returns it as an integer.
`char *gets( char *s );`	Inputs characters from the standard input into the array `s` until a newline or end-of-file character is encountered. A terminating null character is appended to the array.
`int putchar( int c );`	Prints the character stored in `c`.
`int puts( const char *s );`	Prints the string `s` followed by a newline character.
`int sprintf( char *s, const char *format, ... );`	Equivalent to `printf`, except the output is stored in the array `s` instead of printing it on the screen.
`int sscanf( char *s, const char *format, ... );`	Equivalent to `scanf`, except the input is read from the array `s` instead of reading it from the keyboard.

Characters & Strings Example 7

```c
/* Fig. 8.13: fig08_13.c
   Using gets and putchar */
#include <stdio.h>

int main()
{
   char sentence[ 80 ]; /* create char array */

   void reverse( const char * const sPtr ); /* prototype */

    printf( "Enter a line of text:\n" );

    /* use gets to read line of text */
    gets( sentence );

    printf( "\nThe line printed backwards is:\n" );
    reverse( sentence );

    return 0; /* indicates successful termination */

 } /* end main */

```

Characters & Strings Example 7

```
/* recursively outputs characters in string in reverse order */
void reverse( const char * const sPtr )
{
   /* if end of the string */
   if ( sPtr[ 0 ] == '\0' ) {
      return;
   } /* end if */
   else { /* if not end of the string */
      reverse( &sPtr[ 1 ] );

      putchar( sPtr[ 0 ] ); /* use putchar to display character */
   } /* end else */

} /* end function reverse */
```

Program Output:

```
Enter a line of text:
Characters and Strings

The line printed backwards is:
sgnirtS dna sretcarahC

Enter a line of text:
able was I ere I saw elba

The line printed backwards is:
able was I ere I saw elba
```

```c
/* Fig. 8.14: fig08_14.c
   Using getchar and puts */
#include <stdio.h>

int main()
{
   char c;              /* variable to hold character input by user */
   char sentence[ 80 ]; /* create char array */
   int i = 0;           /* initialize counter i */

   /* prompt user to enter line of text */
   puts( "Enter a line of text:" );

   /* use getchar to read each character */
   while ( ( c = getchar() ) != '\n') {
      sentence[ i++ ] = c;
   } /* end while */

   sentence[ i ] = '\0';

   /* use puts to display sentence */
   puts( "\nThe line entered was:" );
   puts( sentence );

   return 0; /* indicates successful termination */

} /* end main */
```

Characters & Strings Example 8

Program Output:

```
Enter a line of text:
This is a test.

The line entered was:
This is a test.
```

Characters & Strings Example 9

```c
/* Fig. 8.15: fig08_15.c
   Using sprintf */
#include <stdio.h>

int main()
{
   char s[ 80 ]; /* create char array */
   int x;        /* define x */
   double y;     /* define y */

    printf( "Enter an integer and a double:\n" );
    scanf( "%d%lf", &x, &y );

    sprintf( s, "integer:%6d\ndouble:%8.2f", x, y );

    printf( "%s\n%s\n",
            "The formatted output stored in array s is:", s );

    return 0; /* indicates successful termination */

} /* end main */
```

Program Output:

```
Enter an integer and a double:
298 87.375
The formatted output stored in array s is:
integer:   298
double:   87.38
```

Characters & Strings Example 10

```c
/* Fig. 8.16: fig08_16.c
   Using sscanf */
#include <stdio.h>

int main()
{
   char s[] = "31298 87.375"; /* initialize array s */
   int x;                     /* define x */
   double y;                  /* define y */

   sscanf( s, "%d%lf", &x, &y );

   printf( "%s\n%s%6d\n%s%8.3f\n",
           "The values stored in character array s are:",
           "integer:", x, "double:", y );

   return 0; /* indicates successful termination */

} /* end main */
```

Program Output:

```
The values stored in character array s are:
integer: 31298
double:  87.375
```

Thank You!

Questions?