Programming Languages -1 (Introduction to C)

strings

Instructor: M.Fatih AMASYALI
E-mail:mfatih@ce.yildiz.edu.tr

The Data Type char

- The data type `char` can be thought of as either a character or an integer. Typically, a `char` has a value 0-255.

```
printf( “%c”, ‘a’ );  /* a is printed */

printf( “%d”, ‘a’ );  /* 97 is printed */

printf( “%c”, 97 );   /* a is printed */
```

```
‘a’ == 97, ‘b’ == 98, …, ‘z’ == 112

‘A’ == 65, ‘B’ == 66, …, ‘Z’ = 90

‘0’ == 48, ‘1’ == 49, …, ’9’ == 57

‘&’ == 38, ‘*’ == 42, …
```

Codes corresponding to characters

- For use inside in single-quotes, or double-quotes, for instance in passing a string to printf

Character	Escape Sequence	Integer Value
Newline	\n	10
Backslash (\)	\\	92
Single quote	\’	39
Double quote	\”	34
Horizontal tab	\t	9
Question Mark	\?	63
Null Character	\0	0

Strings

- Strings are one-dimensional arrays of type `char`. Hence, they have type `char *`.
- By convention, a string in C is terminated by \0 (null character); thus it needs space equal to the size of string +1

C	o	r	n	e	l	l		U	n	i	\0
0	1	2	3	4	5	6	7	8	9	10	11

- A string constant is treated by the compiler as a pointer; also space in memory is allocated for storing the characters.

```
char *p = “abc”;
printf(“%s %s\n”, p, p+1 ); /* prints  abc bc */
printf(“%c”,"abc”[1]); /* prints  b */
printf(“%c”,*(“abc” + 2)); /* prints  c */
```

Example: “Double” printing

```
#include <stdio.h>

void new_print( char *s )
{
  int i;
  for( i = 0; s[i] != 0; i++ )
    {
      printf( "%c%c", s[i], s[i] );
    }
}

void main()
{
  new_print( “Cornell" );
}
```

All of {0 , ‘\0’, NULL} are legal.

Strings are also char pointers

```
#include <stdio.h>

void new_print( char *s )
{
  while (*s)
  {
    printf( "%c%c", *s, *s );
    s++;
  }
}

void main()
{
  new_print( “Cornell" );
}
```

Example: “squeeze” function

```
/* squeeze deletes all instances of c from s */

void squeeze( char s[], int c )
{
  int i, j;

  for( i = j = 0; s[i] != ‘\0’; i++ )
  {
    if( s[i] != c )
    {
      s[j] = s[i];
      j++;
    }
  }
  s[j] = ‘\0’;
}
```

Usage:

```
char p[]="Cornell";
squeeze( p, 'o');
printf("%s\n",p);
```

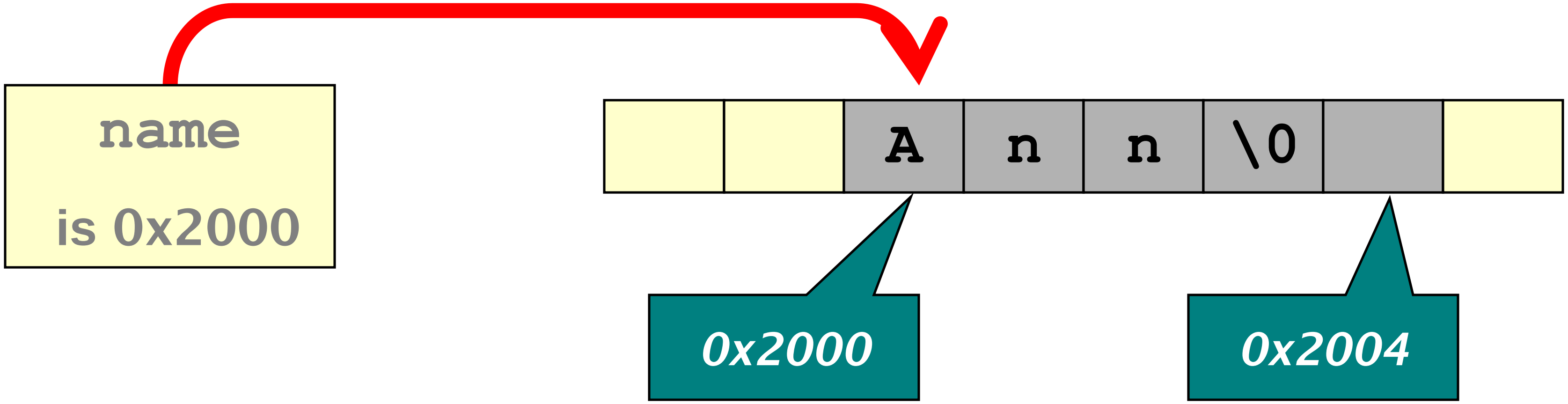
String Input/Output

```
#include <stdio.h>

#define MAXLENGTH 15

int main()
{
   char string1[MAXLENGTH];
   char string2[MAXLENGTH];

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

   return 0;
}
```

Character String Declaration

Declaration 1:

```
char  name[5] = “Ann”;
```



Character String Declaration

Declaration 1:

```
char  name[5] = “Ann”;
```

Could have defined this as an array:

```
char  name[5] = {’A’,’n’,’n’,’\0’};
```

0x2000

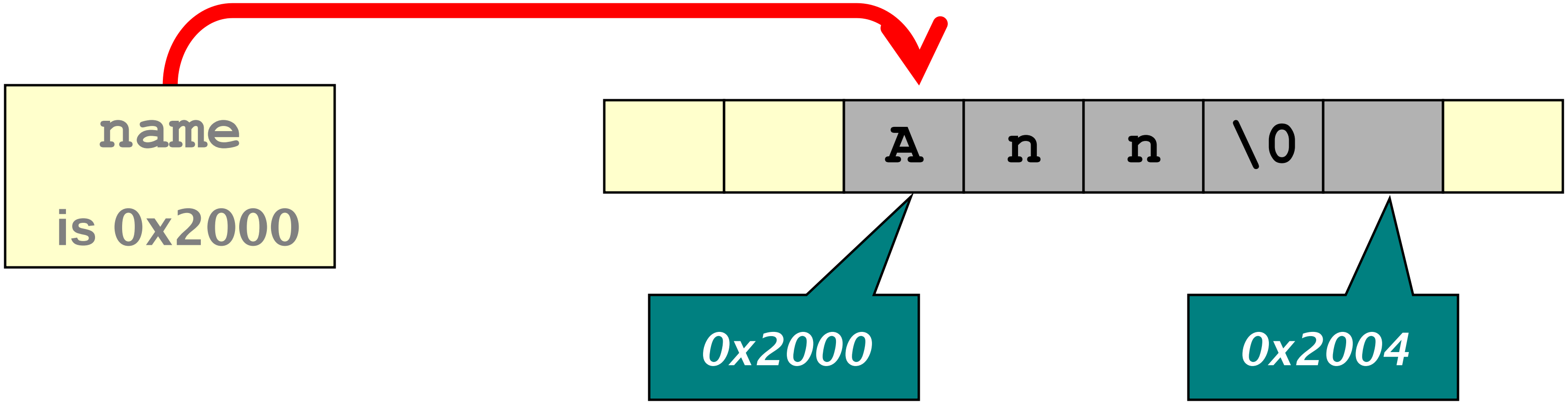
0x2004

Character String Declaration (cont)

Declaration 1:

```
char  name[5] = “Ann”;
```

Can store at most 4 letters, because of `\0’



name

is 0x2000

		A	n	n	\0		

0x2000

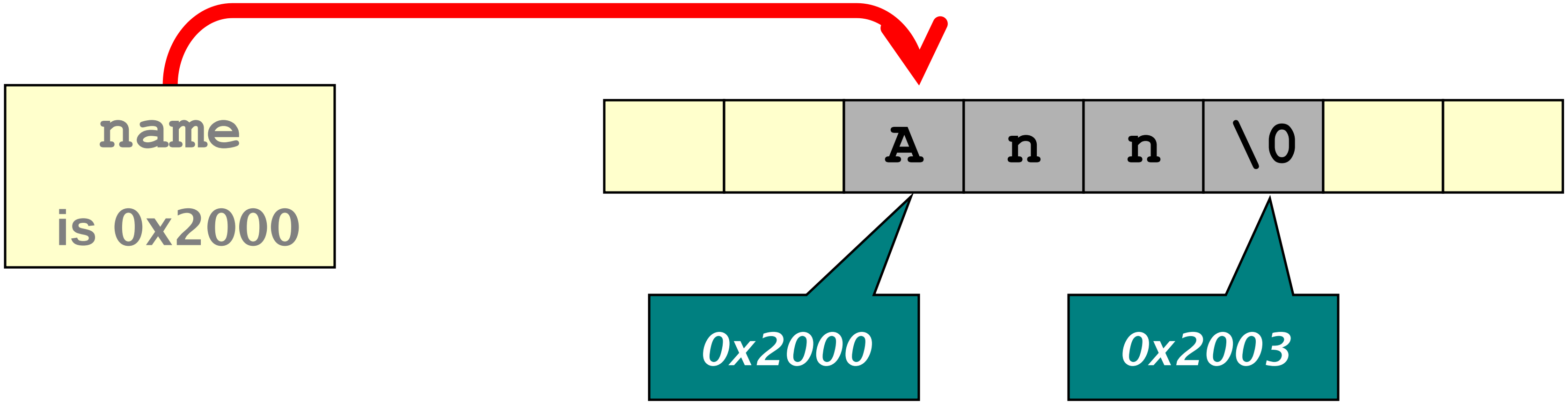
0x2004

Character String Declaration (cont)

Declaration 2: ⚠

```
char  name[] = “Ann”;
```

Takes up an extra cell for ‘\0’



name

is 0x2000

		A	n	n	\0		

0x2000

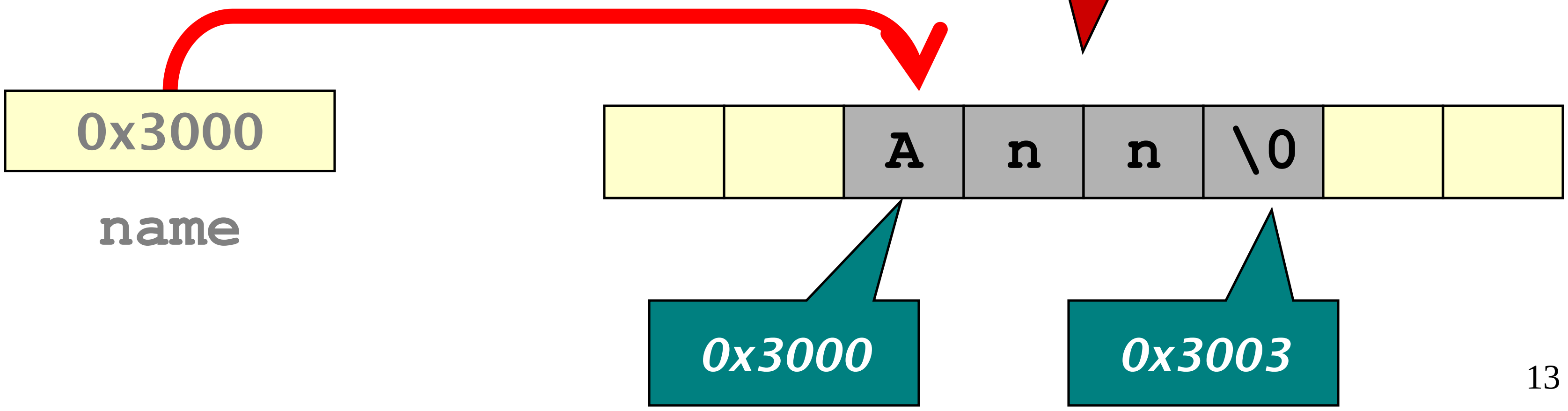
0x2003

Character String Declaration (cont)

Declaration 3: ⚠ ⚠

```
char  *name = “Ann”;
```

Result is “undefined” if you try to modify this string



Character String Declaration (cont)

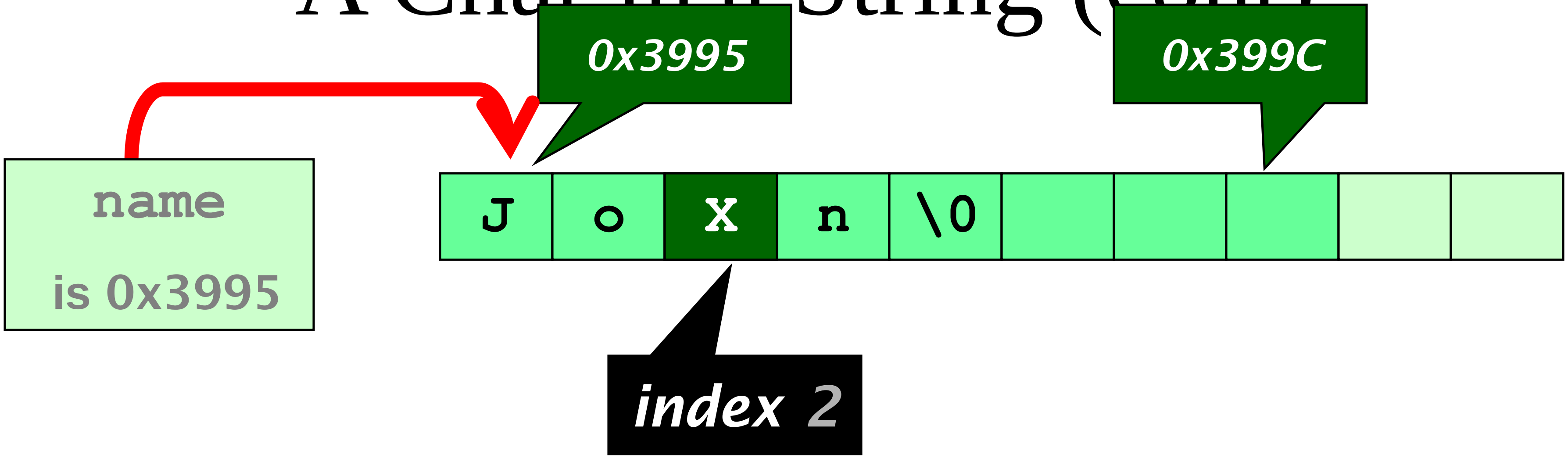
Declaration 4: STOP

```
char   name[];
```

String with arbitrary length?

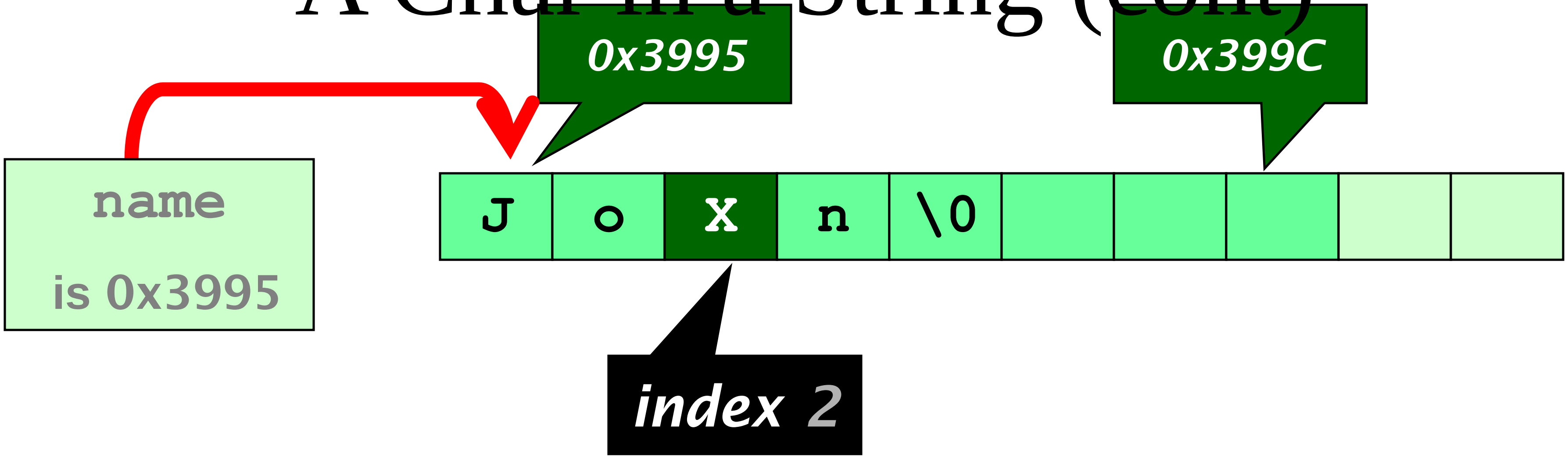
No! Will cause an error "storage size of k isn't known"

A Char in a String (cont)



```
char name[8] = "John";

name[2] = 'X';
printf("Name: %s\n", name);
```

A Char in a String (cont)



name

is 0x3995

0x3995

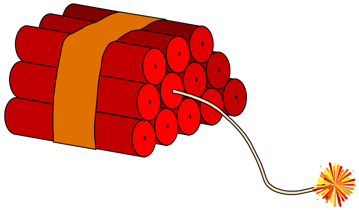
0x399C

J o X n \0

index 2

```
char name[8] = "John";

name[2] = 'X';
printf("Name: %s\n", name);
```

```
output: Name: JoXn
```



Common Mistake 1:

Example:

```
char  name1[5] = “Anne”;
char  name2[5] = “Dave”;

name2 = name1;
```

```
Error: “ISO C++ forbids
assingment of arrays”
```

Caution 1:

Pointer Assignment

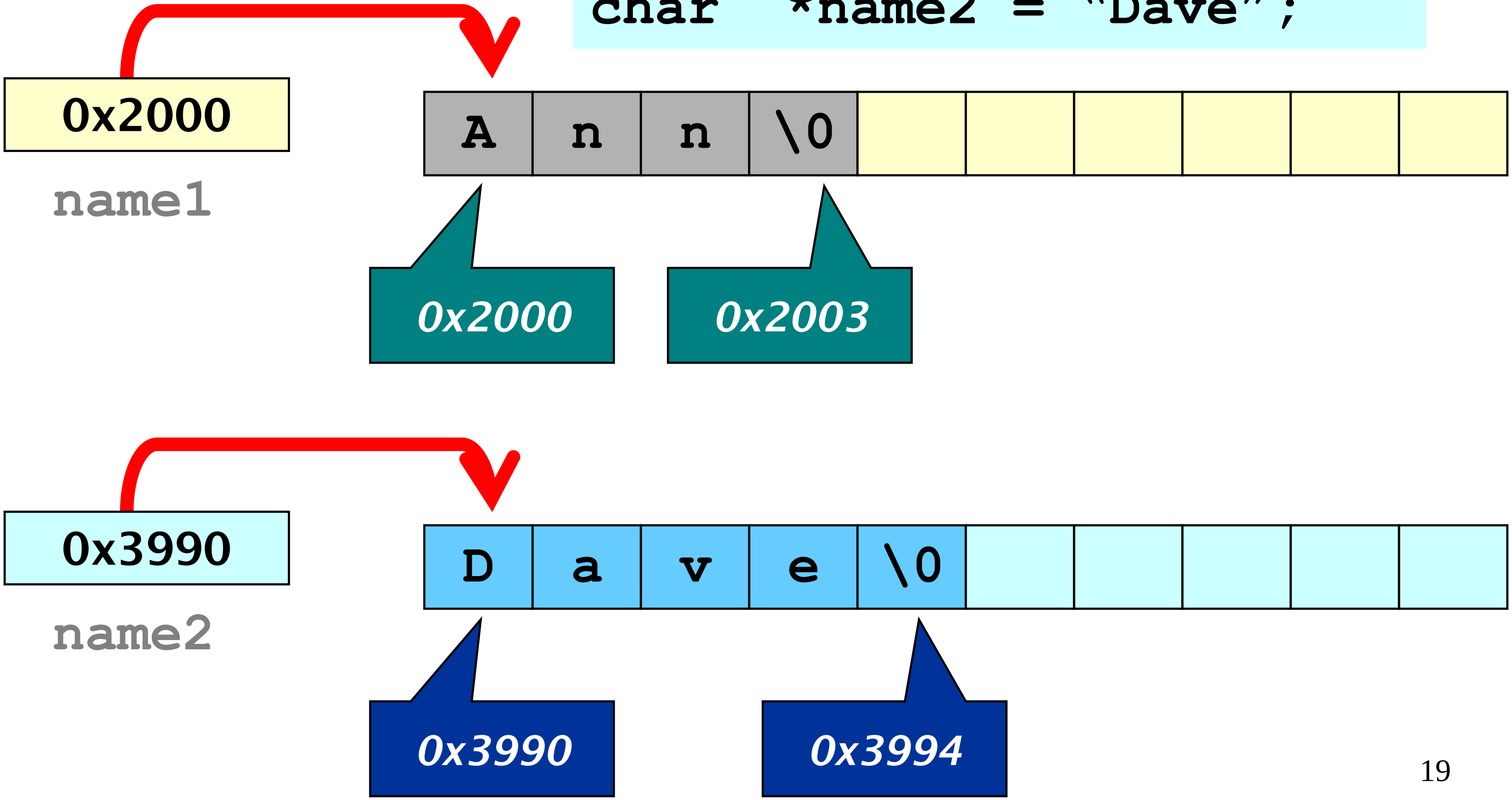
Example:

```
char  *name1 = “Ann”;
char  *name2 = “Dave”;

name2 = name1;
```

Caution 1:

Pointer Assignment

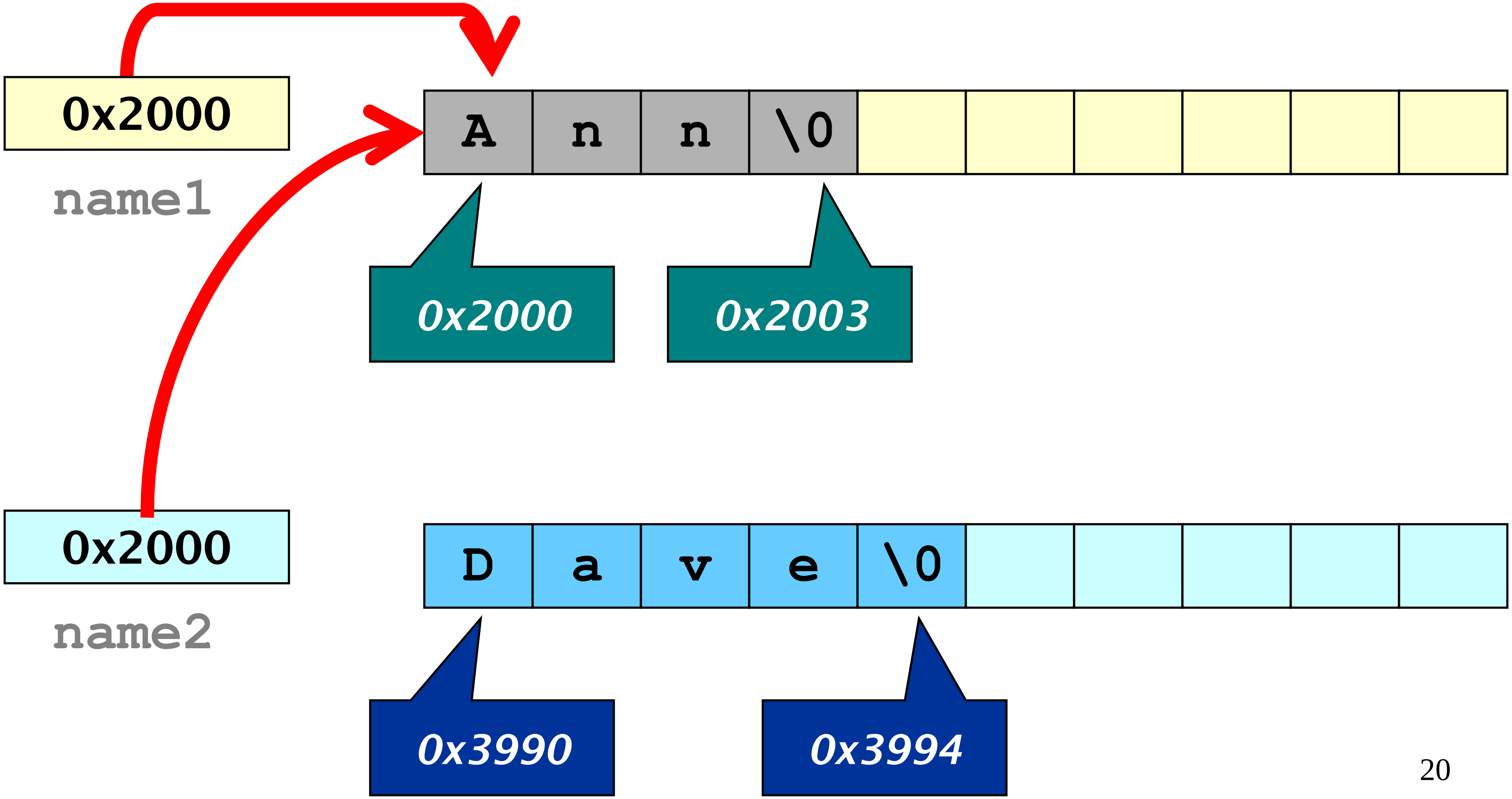
```
char  *name1 = “Ann”;
char  *name2 = “Dave”;
```



Caution 1:

Pointer Assignment

```
name2 = name1;
```



0x2000

name1

A	n	n	\0						

0x2000

0x2003

0x2000

name2

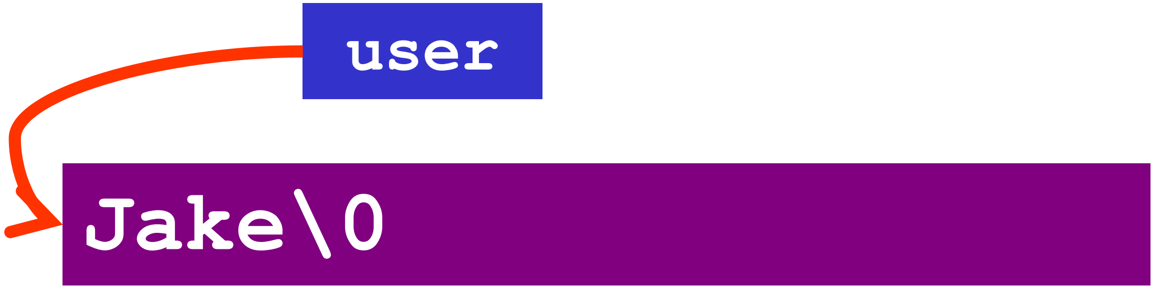
D	a	v	e	\0					

0x3990

0x3994

```
#include <stdio.h>
#include <string.h>
#define  NAMELEN  50

/* Print a simple greeting to
   the user */

void Greet ( char * name )
{
  strcat(name, "! How are ya?");
}
```

```

int main()
{
  char user[NAMELEN];

  printf("Who are you? ");
  scanf("%s", user);
  Greet(user);
  printf("%s\n", user);

  return 0;
}
```

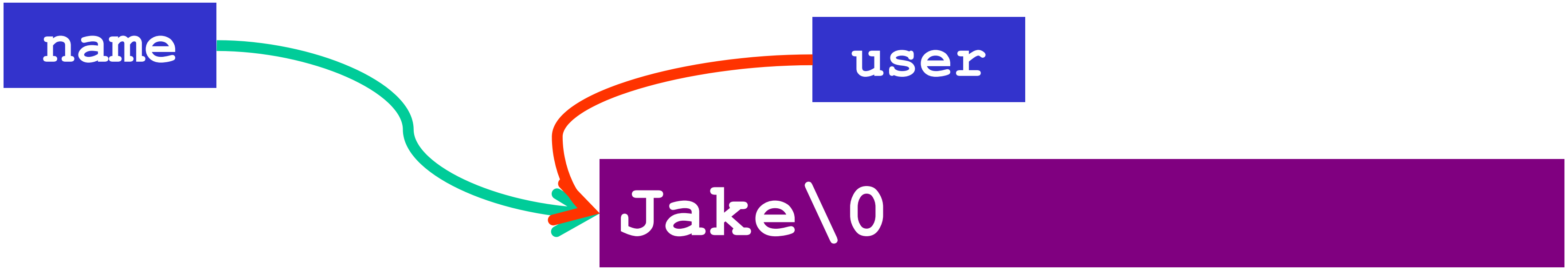


user

Jake\0

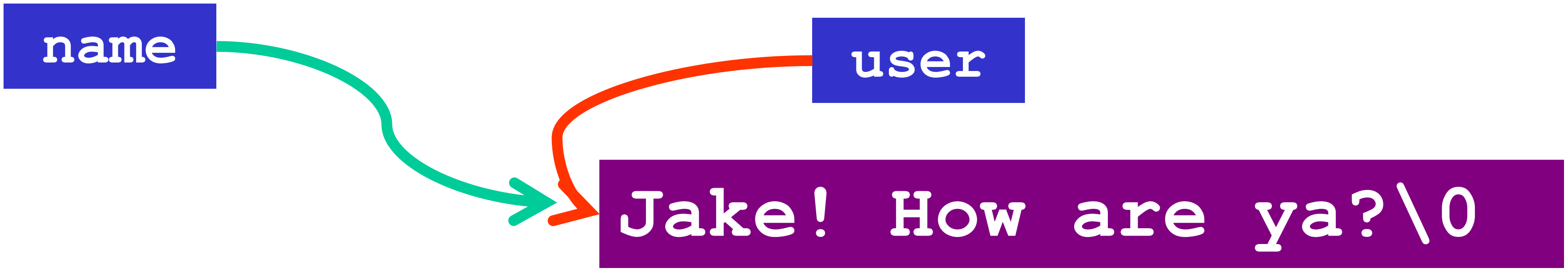
```
#include <stdio.h>
#include <string.h>
#define  NAMELEN  50

/* Print a simple greeting to
   the user */

void Greet ( char * name )
{
  strcat(name, "! How are ya?");
}
```

```

int main()
{
  char user[NAMELEN];

  printf("Who are you? ");
  scanf("%s", user);
  Greet(user);
  printf("%s\n", user);

  return 0;
}
```



```c
#include <stdio.h>
#include <string.h>
#define  NAMELEN  50

/* Print a simple greeting to
   the user */

void Greet ( char * name )
{
  strcat(name, "! How are ya?");
}
```

```c
int main()
{
  char user[NAMELEN];

  printf("Who are you? ");
  scanf("%s", user);
  Greet(user);
  printf("%s\n", user);

  return 0;
}
```



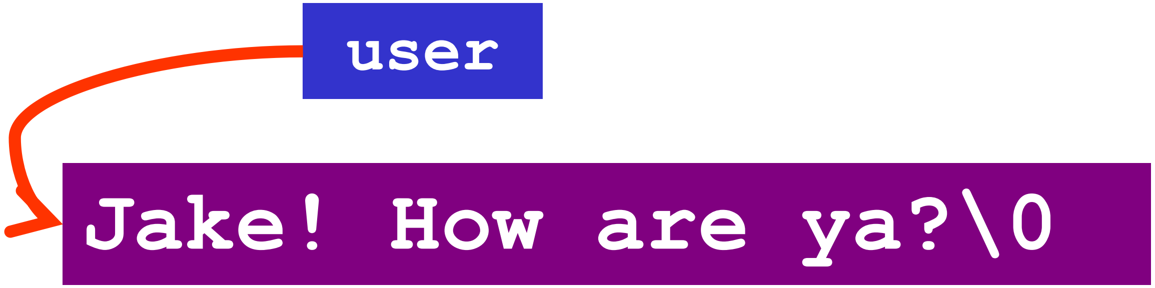
name

user

Jake! How are ya?\0

```c
#include <stdio.h>
#include <string.h>
#define  NAMELEN  50

/* Print a simple greeting to
   the user */

void Greet ( char * name )
{
  strcat(name, "! How are ya?");
}
```

```c
int main()
{
  char user[NAMELEN];

  printf("Who are you? ");
  scanf("%s", user);
  Greet(user);
  printf("%s\n", user);

  return 0;
}
```



user

Jake! How are ya?\0

Prototype	Function description
`int isdigit( int c );`	Returns a true value if `c` is a digit and 0 (false) otherwise.
`int isalpha( int c );`	Returns a true value if `c` is a letter and 0 otherwise.
`int isalnum( int c );`	Returns a true value if `c` is a digit or a letter and 0 otherwise.
`int isxdigit( int c );`	Returns a true value if `c` is a hexadecimal digit character and 0 otherwise. (See Appendix E, Number Systems, for a detailed explanation of binary numbers, octal numbers, decimal numbers and hexadecimal numbers.)
`int islower( int c );`	Returns a true value if `c` is a lowercase letter and 0 otherwise.
`int isupper( int c );`	Returns a true value if `c` is an uppercase letter and 0 otherwise.
`int tolower( int c );`	If `c` is an uppercase letter, `tolower` returns `c` as a lowercase letter. Otherwise, `tolower` returns the argument unchanged.

Character-handling library functions `<ctype.h>`.

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

int main( void )
{
   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" );

```

isdigit tests if a character is a decimal digit

isalpha tests if a character is 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" );

    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 ",
```

`isdigit` tests if a character is a decimal digit or a letter

`isxdigit` tests if a character is a hexadecimal digit

```c
         "hexadecimal digit",
         isxdigit( 'f' ) ? "f is a " : "f is not a ",
         "hexadecimal digit" );

   return 0; /* indicates successful termination */

} /* end main */
```

```
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( void )
{
   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" );

```

`islower` tests if a character is a lowercase letter

`isupper` tests if a character is an uppercase letter

```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 */
```

`toupper` and **`tolower`** convert letters to upper or lower case

```
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
```

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`.
`double strtod( const char *nPtr, char **endPtr );`	Converts the string `nPtr` to `double`.
`long strtol( const char *nPtr, char **endPtr, int base );`	Converts the string `nPtr` to `long`.
`unsigned long strtoul( const char *nPtr, char **endPtr, int base );`	Converts the string `nPtr` to `unsigned long`.

String-conversion functions of the general utilities library `<stdlib.h>`.

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

int main( void )
{
   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 */
```

atof converts a string to a **double**

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

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

int main( void )
{
   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 */
```

atoi converts a string to an **int**

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

```c
/* Fig. 8.9: fig08_09.c
   Using strtod */
#include <stdio.h>
#include <stdlib.h>

int main( void )
{
   /* initialize string pointer */
   const char *string = "51.2% are admitted"; /* initialize string */

   double d;        /* variable to hold converted sequence */
   char *stringPtr; /* create char pointer */

   d = strtod( string, &stringPtr );

   printf( "The string \"%s\" is converted to the\n", string );
   printf( "double value %.2f and the string \"%s\"\n", d, stringPtr );

   return 0; /* indicates successful termination */

} /* end main */
```

strtod converts a piece of a string to a **double**

```
The string "51.2% are admitted" is converted to the
double value 51.20 and the string "% are admitted"
```

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. Returns the string inputted into `s`. Note that an error will occur if `s` is not large enough to hold the string.
`int putchar( int c );`	Prints the character stored in `c` and returns it as an integer.
`int puts( const char *s );`	Prints the string `s` followed by a newline character. Returns a non-zero integer if successful, or `EOF` if an error occurs.
`int sprintf( char *s, const char *format, ... );`	Equivalent to `printf`, except the output is stored in the array `s` instead of printed on the screen. Returns the number of characters written to `s`, or `EOF` if an error occurs.
`int sscanf( char *s, const char *format, ... );`	Equivalent to `scanf`, except the input is read from the array `s` rather than from the keyboard. Returns the number of items successfully read by the function, or `EOF` if an error occurs.

Standard input/output library character and string functions.

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

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

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

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

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

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

   return 0; /* indicates successful termination */

} /* end main */
```

`gets` reads a line of text from the user

```c

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

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

} /* end function reverse */
```

putchar prints a single character on the screen

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

The line printed backward is:
sgnirtS dna sretcarahC
```

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

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

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

int main( void )
{
   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'; /* terminate string */

```

puts prints a line of text on the screen

getchar reads a single character from the user

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

   return 0; /* indicates successful termination */

} /* end main */
```

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

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

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

int main( void )
{
   char s[ 80 ]; /* create char array */
   int x;        /* x value to be input */
   double y;     /* y value to be input */

   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 */
```

sprintf prints a line of text into an array like **printf** prints text on the screen

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

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

int main( void )
{
   char s[] = "31298 87.375"; /* initialize array s */
   int x;    /* x value to be input */
   double y; /* y value to be input */

   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 */
```

`sscanf` reads a line of text from an array like **`scanf`** reads text from the user

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

Function prototype	Function description
`char *strcpy( char *s1, const char *s2 )`	Copies string `s2` into array `s1`. The value of `s1` is returned.
`char *strncpy( char *s1, const char *s2, size_t n )`	Copies at most `n` characters of string `s2` into array `s1`. The value of `s1` is returned.
`char *strcat( char *s1, const char *s2 )`	Appends string `s2` to array `s1`. The first character of `s2` overwrites the terminating null character of `s1`. The value of `s1` is returned.
`char *strncat( char *s1, const char *s2, size_t n )`	Appends at most `n` characters of string `s2` to array `s1`. The first character of `s2` overwrites the terminating null character of `s1`. The value of `s1` is returned.

String-manipulation functions of the string-handling library.

Portability Tip

- Type `size_t` is a system-dependent synonym for either type `unsigned long` or type `unsigned int`.

```c
/* Fig. 8.18: fig08_18.c
   Using strcpy and strncpy */
#include <stdio.h>
#include <string.h>

int main( void )
{
   char x[] = "Happy Birthday to You"; /* initialize char array x */
   char y[ 25 ]; /* create char array y */
   char z[ 15 ]; /* create char array z */

   /* copy contents of x into y */
   printf( "%s%s\n%s%s\n",
      "The string in array x is: ", x,
      "The string in array y is: ", strcpy( y, x ) );

   /* copy first 14 characters of x into z. Does not copy null
      character */
   strncpy( z, x, 14 );

   z[ 14 ] = '\0'; /* terminate string in z */
   printf( "The string in array z is: %s\n", z );

   return 0; /* indicates successful termination */

} /* end main */
```

strcpy copies string **x** into character array **y**

strncpy copies 14 characters of string **x** into character array **z**

Note that **strncpy** does not automatically append a null character

```
The string in array x is: Happy Birthday to You
The string in array y is: Happy Birthday to You
The string in array z is: Happy Birthday
```

```c
/* Fig. 8.19: fig08_19.c
   Using strcat and strncat */
#include <stdio.h>
#include <string.h>

int main( void )
{
   char s1[ 20 ] = "Happy "; /* initialize char array s1 */
   char s2[] = "New Year ";  /* initialize char array s2 */
   char s3[ 40 ] = "";       /* initialize char array s3 to empty */

   printf( "s1 = %s\ns2 = %s\n", s1, s2 );

   /* concatenate s2 to s1 */
   printf( "strcat( s1, s2 ) = %s\n", strcat( s1, s2 ) );

   /* concatenate first 6 characters of s1 to s3. Place '\0'
      after last character */
   printf( "strncat( s3, s1, 6 ) = %s\n", strncat( s3, s1, 6 ) );

```

`strcat` adds the characters of string **`s2`** to the end of string **`s1`**

`strncat` adds the first 6 characters of string **`s1`** to the end of string **`s3`**

```c
   /* concatenate s1 to s3 */
   printf( "strcat( s3, s1 ) = %s\n", strcat( s3, s1 ) );

   return 0; /* indicates successful termination */

} /* end main */
```

```
s1 = Happy
s2 = New Year
strcat( s1, s2 ) = Happy New Year
strncat( s3, s1, 6 ) = Happy
strcat( s3, s1 ) = Happy Happy New Year
```

Comparison Functions of the String-Handling Library

- Comparing strings
  - Computer compares numeric ASCII codes of characters in string

Function prototype	Function description
`int strcmp( const char *s1, const char *s2 );`	Compares the string `s1` with the string `s2`. The function returns `0`, less than `0` or greater than `0` if `s1` is equal to, less than or greater than `s2`, respectively.
`int strncmp( const char *s1, const char *s2, size_t n );`	Compares up to `n` characters of the string `s1` with the string `s2`. The function returns `0`, less than `0` or greater than `0` if `s1` is equal to, less than or greater than `s2`, respectively.

String-comparison functions of the string-handling library.

```c
/* Fig. 8.21: fig08_21.c
   Using strcmp and strncmp */
#include <stdio.h>
#include <string.h>

int main( void )
{
   const char *s1 = "Happy New Year"; /* initialize char pointer */
   const char *s2 = "Happy New Year"; /* initialize char pointer */
   const char *s3 = "Happy Holidays"; /* initialize char pointer */

   printf("%s%s\n%s%s\n%s%s\n\n%s%2d\n%s%2d\n%s%2d\n\n",
          "s1 = ", s1, "s2 = ", s2, "s3 = ", s3,
          "strcmp(s1, s2) = ", strcmp( s1, s2 ),
          "strcmp(s1, s3) = ", strcmp( s1, s3 ),
          "strcmp(s3, s1) = ", strcmp( s3, s1 ) );

```

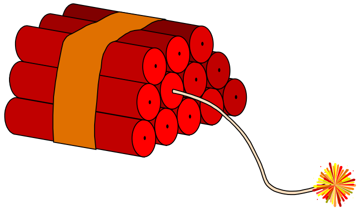
`strcmp` compares string **`s1`** to string **`s2`**

```c
   printf("%s%2d\n%s%2d\n%s%2d\n",
          "strncmp(s1, s3, 6) = ", strncmp( s1, s3, 6 ),
          "strncmp(s1, s3, 7) = ", strncmp( s1, s3, 7 ),
          "strncmp(s3, s1, 7) = ", strncmp( s3, s1, 7 ) );

   return 0; /* indicates successful termination */

} /* end main */
```

```
s1 = Happy New Year
s2 = Happy New Year
s3 = Happy Holidays

strcmp(s1, s2) =  0
strcmp(s1, s3) =  1
strcmp(s3, s1) = -1
```

```
strncmp(s1, s3, 6) =  0
strncmp(s1, s3, 7) =  1
strncmp(s3, s1, 7) = -1
```

strncmp compares the first 6 characters of string **s1** to the first **X** characters of string **s3**



Common Mistake:

Wrong Comparison

✗

```
strcpy(string1, “Apple”);
strcpy(string2, “Wax”);

if (string1 < string2)
{
  printf(“%s %s\n”, string1, string2);
}
else
{
  printf(“%s %s\n”, string2, string1);
}
```

Caution 1:

Not a Boolean

```
strcpy(string1, “Hi Mum”);
strcpy(string2, “Hi Mum”);

if ( strcmp(string1, string2) )
{
  printf(“%s and %s are the same\n”,
        string1, string2);
}
```

Returns zero if the strings are the same.

if (strcmp(string1, string2) == 0)

String Operation: Length

```
char string1[100];

strcpy(string1, “Apple”);

printf(“%d\n”, strlen(string1));
```

```
output: 5
```

Number of char-s before the `\0’

Function prototype	Function description
`char *strchr( const char *s, int c );`	Locates the first occurrence of character `c` in string `s`. If `c` is found, a pointer to `c` in `s` is returned. Otherwise, a `NULL` pointer is returned.
`size_t strcspn( const char *s1, const char *s2 );`	Determines and returns the length of the initial segment of string `s1` consisting of characters not contained in string `s2`.
`size_t strspn( const char *s1, const char *s2 );`	Determines and returns the length of the initial segment of string `s1` consisting only of characters contained in string `s2`.
`char *strpbrk( const char *s1, const char *s2 );`	Locates the first occurrence in string `s1` of any character in string `s2`. If a character from string `s2` is found, a pointer to the character in string `s1` is returned. Otherwise, a `NULL` pointer is returned.

String-manipulation functions of the string-handling library.

Function prototype	Function description
`char *strrchr( const char *s, int c );`	Locates the last occurrence of `c` in string `s`. If `c` is found, a pointer to `c` in string `s` is returned. Otherwise, a `NULL` pointer is returned.
`char *strstr( const char *s1, const char *s2 );`	Locates the first occurrence in string `s1` of string `s2`. If the string is found, a pointer to the string in `s1` is returned. Otherwise, a `NULL` pointer is returned.
`char *strtok( char *s1, const char *s2 );`	A sequence of calls to `strtok` breaks string `s1` into “tokens”—logical pieces such as words in a line of text—separated by characters contained in string `s2`. The first call contains `s1` as the first argument, and subsequent calls to continue tokenizing the same string contain `NULL` as the first argument. A pointer to the current token is returned by each call. If there are no more tokens when the function is called, `NULL` is returned.

String-manipulation functions of the string-handling library.

```c
/* Fig. 8.23: fig08_23.c
   Using strchr */
#include <stdio.h>
#include <string.h>

int main( void )
{
   const char *string = "This is a test"; /* initialize char pointer */
   char character1 = 'a'; /* initialize character1 */
   char character2 = 'z'; /* initialize character2 */

   /* if character1 was found in string */
   if ( strchr( string, character1 ) != NULL ) {
      printf( "\'%c\' was found in \"%s\".\n",
         character1, string );
   } /* end if */
   else { /* if character1 was not found */
      printf( "\'%c\' was not found in \"%s\".\n",
         character1, string );
   } /* end else */
```

strchr searches for the first instance of **character1** in **string**

```c

   /* if character2 was found in string */
   if ( strchr( string, character2 ) != NULL ) {
      printf( "\'%c\' was found in \"%s\".\n",
         character2, string );
   } /* end if */
   else { /* if character2 was not found */
      printf( "\'%c\' was not found in \"%s\".\n",
         character2, string );
   } /* end else */

   return 0; /* indicates successful termination */

} /* end main */
```

```
'a' was found in "This is a test".
'z' was not found in "This is a test".
```

```c
/* Fig. 8.24: fig08_24.c
   Using strcspn */
#include <stdio.h>
#include <string.h>

int main( void )
{
   /* initialize two char pointers */
   const char *string1 = "The value is 3.14159";
   const char *string2 = "1234567890";

   printf( "%s%s\n%s%s\n\n%s\n%s%u\n",
      "string1 = ", string1, "string2 = ", string2,
      "The length of the initial segment of string1",
      "containing no characters from string2 = ",
      strcspn( string1, string2 ) );

   return 0; /* indicates successful termination */

} /* end main */
```

strcspn returns the length of the initial segment of **string1** that does not contain any characters in **string2**

```
string1 = The value is 3.14159
string2 = 1234567890

The length of the initial segment of string1
containing no characters from string2 = 13
```

String1'in,
String2'deki karakterlerden
hiçbirini içermeyen,
ilk kısmının uzunluğu

```c
/* Fig. 8.25: fig08_25.c
   Using strpbrk */
#include <stdio.h>
#include <string.h>

int main( void )
{
   const char *string1 = "This is a test"; /* initialize char pointer */
   const char *string2 = "beware";         /* initialize char pointer */

   printf( "%s\"%s\"\n'%c'%s\n\"%s\"\n",
      "Of the characters in ", string2,
      *strpbrk( string1, string2 ),
      " appears earliest in ", string1 );

   return 0; /* indicates successful termination */

} /* end main */
```

```
Of the characters in "beware"
'a' appears earliest in
"This is a test"
```

strpbrk returns a pointer to the first appearance in **string1** of any character from **string2**

String1’de,
String2’deki karakterlerden
herhangi birinin ilk geçtiği yerin
adresi

```c
/* Fig. 8.26: fig08_26.c
   Using strrchr */
#include <stdio.h>
#include <string.h>

int main( void )
{
   /* initialize char pointer */
   const char *string1 = "A zoo has many animals including zebras";

   int c = 'z'; /* character to search for */

   printf( "%s\n%s'%c'%s\"%s\"\n",
           "The remainder of string1 beginning with the",
           "last occurrence of character ", c,
           " is: ", strrchr( string1, c ) );

   return 0; /* indicates successful termination */

} /* end main */
```

strrchr returns the remainder of **string1** following the last occurrence of the character **c**

```
The remainder of string1 beginning with the
last occurrence of character 'z' is: "zebras"
```

String1’in, c karakterinin son geçtiği yerden itibarenki kısmı

```c
/* Fig. 8.27: fig08_27.c
   Using strspn */
#include <stdio.h>
#include <string.h>

int main( void )
{
   /* initialize two char pointers */
   const char *string1 = "The value is 3.14159";
   const char *string2 = "aehi lsTuv";

   printf( "%s%s\n%s%s\n\n%s\n%s%u\n",
      "string1 = ", string1, "string2 = ", string2,
      "The length of the initial segment of string1",
      "containing only characters from string2 = ",
      strspn( string1, string2 ) );

   return 0; /* indicates successful termination */

} /* end main */
```

strspn returns the length of the initial segment of **string1** that contains only characters from **string2**

```
string1 = The value is 3.14159
string2 = aehi lsTuv

The length of the initial segment of string1
containing only characters from string2 = 13
```

String1’in,
Sadece String2’deki karakterlerden
oluşan ilk kısmının uzunluğu

```c
/* Fig. 8.28: fig08_28.c
   Using strstr */
#include <stdio.h>
#include <string.h>

int main( void )
{
   const char *string1 = "abcdefabcdef"; /* string to search */
   const char *string2 = "def"; /* string to search for */

   printf( "%s%s\n%s%s\n\n%s\n%s%s\n",
      "string1 = ", string1, "string2 = ", string2,
      "The remainder of string1 beginning with the",
      "first occurrence of string2 is: ",
      strstr( string1, string2 ) );

   return 0; /* indicates successful termination

} /* end main */
```

strstr returns the remainder of **string1** following the first occurrence of **string2**

```
string1 = abcdefabcdef
string2 = def

The remainder of string1 beginning with the
first occurrence of string2 is: defabcdef
```

String1'in, string2'nin ilk geçtiği yerden itibarenki kısmı

```c
/* Fig. 8.29: fig08_29.c
   Using strtok */
#include <stdio.h>
#include <string.h>

int main( void )
{
   /* initialize array string */
   char string[] = "This is a sentence with 7 tokens";
   char *tokenPtr; /* create char pointer */

   printf( "%s\n%s\n\n%s\n",
      "The string to be tokenized is:", string,
      "The tokens are:" );

   tokenPtr = strtok( string, " " ); /* begin tokenizing sentence */

   /* continue tokenizing sentence until tokenPtr becomes NULL */
   while ( tokenPtr != NULL ) {
      printf( "%s\n", tokenPtr );
      tokenPtr = strtok( NULL, " " ); /* get next token */
   } /* end while */
```

`strtok` “tokenizes” **`string`** by breaking it into tokens at each space

Calling **`strtok`** again and passing it **`NULL`** continues the tokenizing of the previous string

```c

   return 0; /* indicates successful termination */

} /* end main */
```

```
The string to be tokenized is:
This is a sentence with 7 tokens

The tokens are:
This
is
a
sentence
with
7
tokens
```

Memory Functions of the String-Handling Library

- Memory Functions
  - In `<stdlib.h>`
  - Manipulate, compare, and search blocks of memory
  - Can manipulate any block of data
- Pointer parameters are `void *`
  - Any pointer can be assigned to `void *`, and vice versa
  - `void *` cannot be dereferenced
    - Each function receives a size argument specifying the number of bytes (characters) to process

Function prototype	Function description
void *memcpy(void *s1, const void *s2, size_t n);	Copies n characters from the object pointed to by s2 into the object pointed to by s1. A pointer to the resulting object is returned.
void *memmove(void *s1, const void *s2, size_t n);	Copies n characters from the object pointed to by s2 into the object pointed to by s1. The copy is performed as if the characters were first copied from the object pointed to by s2 into a temporary array and then from the temporary array into the object pointed to by s1. A pointer to the resulting object is returned.
int memcmp(const void *s1, const void *s2, size_t n);	Compares the first n characters of the objects pointed to by s1 and s2. The function returns 0, less than 0 or greater than 0 if s1 is equal to, less than or greater than s2.
void *memchr(const void *s, int c, size_t n);	Locates the first occurrence of c (converted to unsigned char) in the first n characters of the object pointed to by s. If c is found, a pointer to c in the object is returned. Otherwise, NULL is returned.
void *memset(void *s, int c, size_t n);	Copies c (converted to unsigned char) into the first n characters of the object pointed to by s. A pointer to the result is returned.

Memory functions of the string-handling library.

Common Programming Error 8.8

- String-manipulation functions other than `memmove` that copy characters have undefined results when copying takes place between parts of the same string.

```c
/* Fig. 8.31: fig08_31.c
   Using memcpy */
#include <stdio.h>
#include <string.h>

int main( void )
{
   char s1[ 17 ];                   /* create char array s1 */
   char s2[]  = "Copy this string"; /* initialize char array s2 */

   memcpy( s1, s2, 17 );
   printf( "%s\n%s\"%s\"\n",
           "After s2 is copied into s1 with memcpy,",
           "s1 contains ", s1 );

   return 0; /* indicates successful termination */

} /* end main */
```

memcpy copies the first 17 characters from object **s2** into object **s1**

```
After s2 is copied into s1 with memcpy,
s1 contains "Copy this string"
```

```c
/* Fig. 8.32: fig08_32.c
   Using memmove */
#include <stdio.h>
#include <string.h>

int main( void )
{
   char x[] = "Home Sweet Home"; /* initialize char array x */

   printf( "%s%s\n", "The string in array x before memmove is: ", x );
   printf( "%s%s\n", "The string in array x after memmove is: ",
           memmove( x, &x[ 5 ], 10 ) );

   return 0; /* indicates successful termination */

} /* end main */
```

```
The string in array x before memmove is: Home Sweet Home
The string in array x after memmove is: Sweet Home Home
```

`memmove` copies the first 10 characters from **`x[5]`** into object **`x`** by means of a temporary array

```c
/* Fig. 8.33: fig08_33.c
   Using memcmp */
#include <stdio.h>
#include <string.h>

int main( void )
{
   char s1[] = "ABCDEFG"; /* initialize char array s1 */
   char s2[] = "ABCDXYZ"; /* initialize char array s2 */

   printf( "%s%s\n%s%s\n\n%s%2d\n%s%2d\n%s%2d\n",
           "s1 = ", s1, "s2 = ", s2,
           "memcmp( s1, s2, 4 ) = ", memcmp( s1, s2, 4 ),
           "memcmp( s1, s2, 7 ) = ", memcmp( s1, s2, 7 ),
           "memcmp( s2, s1, 7 ) = ", memcmp( s2, s1, 7 ) );

   return 0; /* indicate successful termination */

} /* end main */
```

memcmp compares the first 4 characters of objects **s1** and **s2**

```
s1 = ABCDEFG
s2 = ABCDXYZ

memcmp( s1, s2, 4 ) =  0
memcmp( s1, s2, 7 ) = -1
memcmp( s2, s1, 7 ) =  1
```

```c
/* Fig. 8.34: fig08_34.c
   Using memchr */
#include <stdio.h>
#include <string.h>

int main( void )
{
   const char *s = "This is a string"; /* initialize char pointer */

   printf( "%s\'%c\'%s\"%s\"\n",
           "The remainder of s after character ", 'r',
           " is found is ", memchr( s, 'r', 16 ) );

   return 0; /* indicates successful termination */

} /* end main */
```

```
The remainder of s after character 'r' is found is "ring"
```

`memchr` locates the first occurrence of the character **`r`** inside the first 16 characters of object **`s`**

```c
/* Fig. 8.35: fig08_35.c
   Using memset */
#include <stdio.h>
#include <string.h>

int main( void )
{
   char string1[ 15 ] = "BBBBBBBBBBBBBB"; /* initialize string1 */

   printf( "string1 = %s\n", string1 );
   printf( "string1 after memset = %s\n", memset( string1, 'b', 7 ) );

   return 0; /* indicates successful termination */

} /* end main */
```

```
string1 = BBBBBBBBBBBBBB
string1 after memset = bbbbbbbBBBBBBB
```

`memset` copies the character **`b`** into the first 7 characters of object **`string1`**

Referance

- Ioannis A. Vetsikas, Lecture notes
- Dale Roberts, Lecture notes