1806ICT
Programming Fundamentals

Strings

1

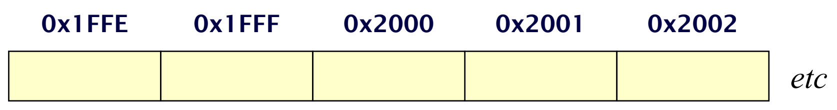
Topics

- Strings Representation
- Strings Declaration
- Index of a char in a String
- String Operations
- Common Mistakes
- Character Testing & Converting Functions
- Arrays of Strings

2

Strings Representation

- Main memory
  - contiguous array of cells
  - each cell has an address



3

3

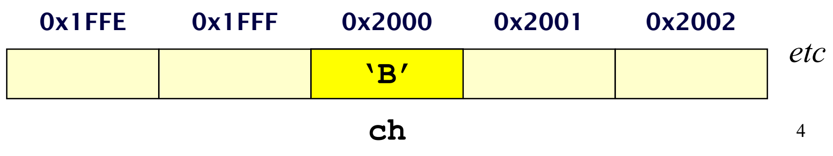
Strings Representation (cont.)

- *Recall:* Variable declaration
  - sets aside a memory location to contain a value

Example:

```
char  ch;
```

```
ch = 'B';
```



4

4

Strings Representation (cont.)

- String declaration
  - sets aside an array of cells
  - each cell contains a char
  - address of first cell in the array

Example: `char  name[5];`

Specifies number of cells in the array

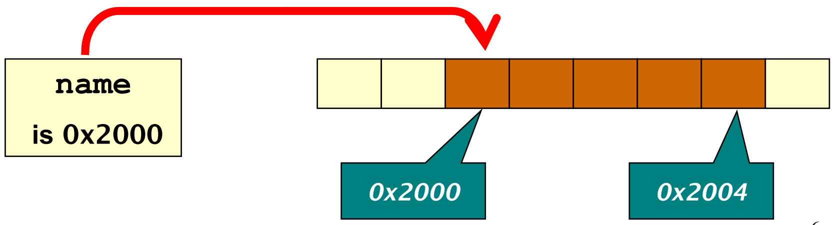
5

5

Strings Representation (cont.)

- String declaration
  - sets aside an array of cells
  - each cell contains a char
  - address of first cell in the array

Example: `char  name[5];`



6

6

Character Arrays vs Character Strings

- A character string is a char array
- A character string *must* have the terminating character (`'\0'`)
- The terminating character allows scanf() and printf() to handle character strings

7

7

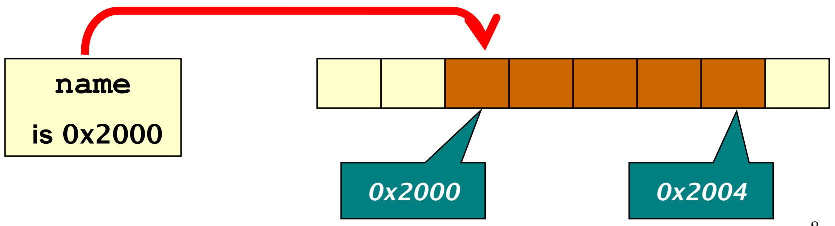
Character Strings

Declaration 1:

```
char  name[5];
```

Declaration 2:

```
#define MAXLENGTH 5

char  name[MAXLENGTH];
```



8

8

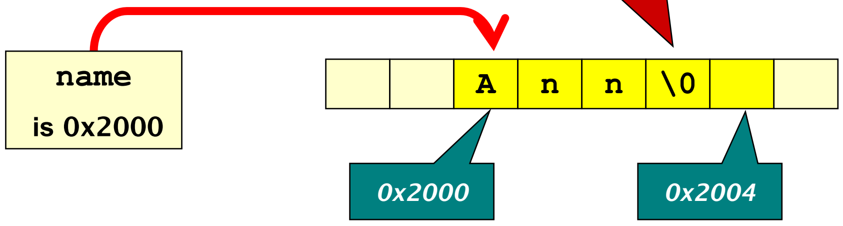
Character String Declaration

Declaration 1:

```
char  name[5] = "Ann";
```

Terminating Character:

- *Marks the end of string*
- *Special char: '\0'*
- *aka NUL (single L)*



9

9

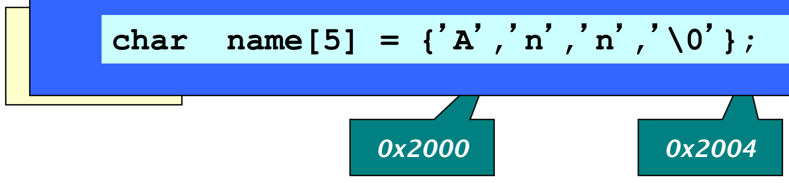
Character String Declaration

Declaration 1:

```
char  name[5] = "Ann";
```

Could have defined this as an array:

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



10

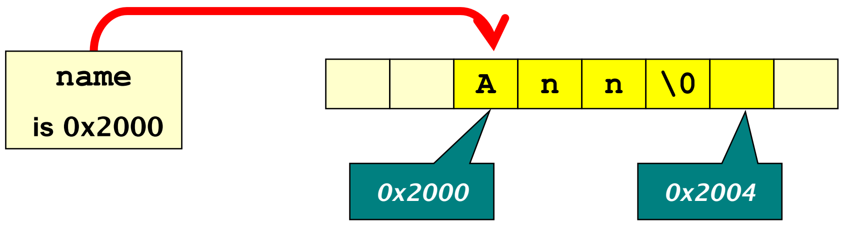
10

Character String Declaration (cont.)

Declaration 1:

```
char  name[5] = "Ann";
```

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



11

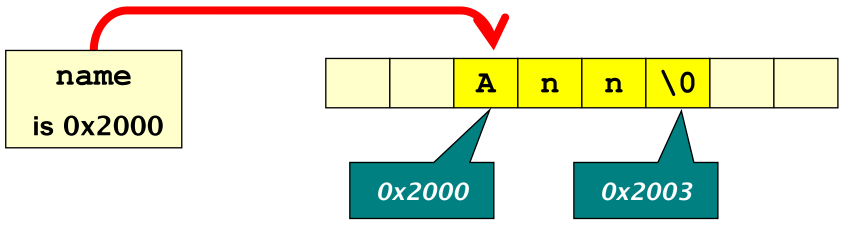
11

Character String Declaration (cont.)

Declaration 2: ⚠

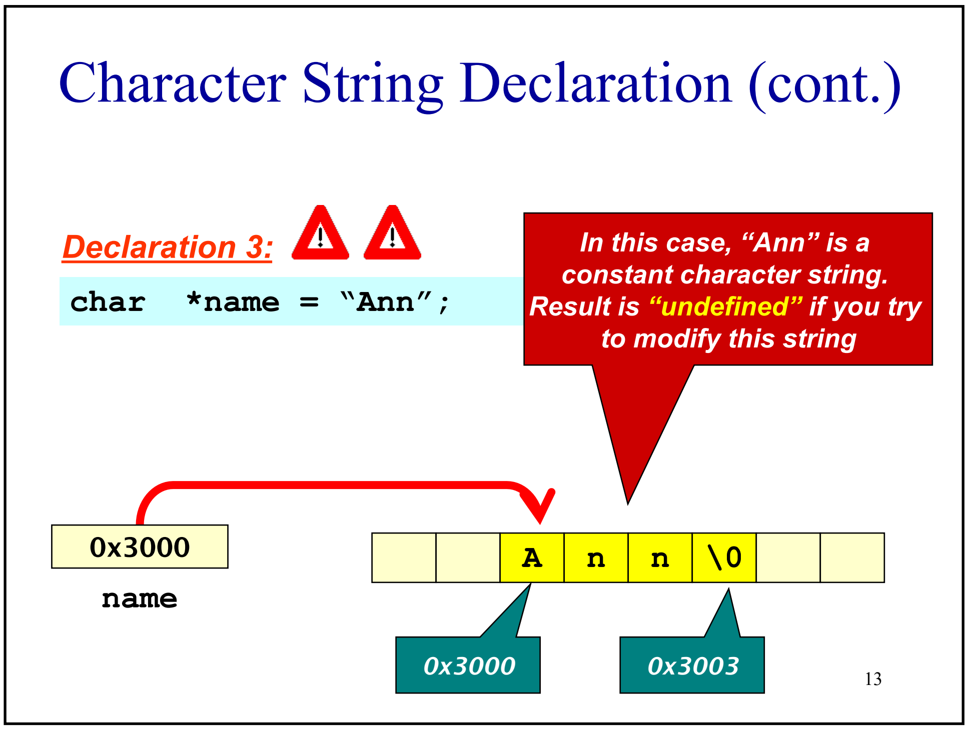
```
char  name[] = "Ann";
```

Takes up an extra cell for '\0'



12

12



Character String Declaration (cont.)

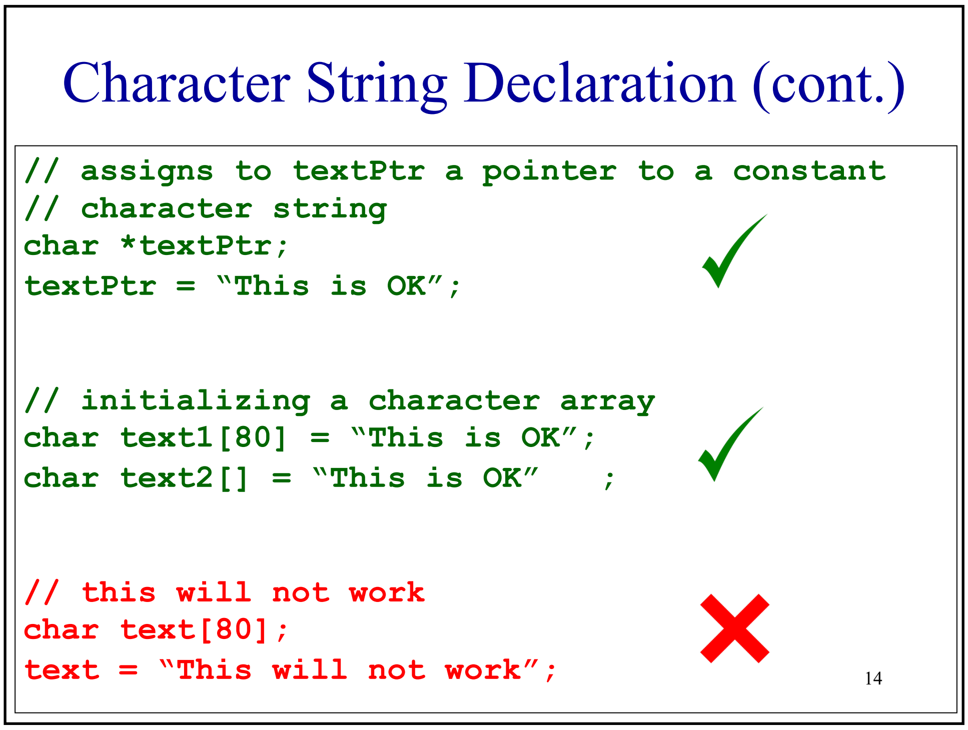
Declaration 3: ⚠ ⚠

```
char  *name = "Ann";
```

In this case, "Ann" is a constant character string. Result is "undefined" if you try to modify this string

13

13



Character String Declaration (cont.)

```
// assigns to textPtr a pointer to a constant
// character string
char *textPtr;
textPtr = "This is OK";
```

✓

```
// initializing a character array
char text1[80] = "This is OK";
char text2[] = "This is OK"   ;
```

✓

```
// this will not work
char text[80];
text = "This will not work";
```

✗

14

14

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;
}
```

No ampersand (&)!

15

15

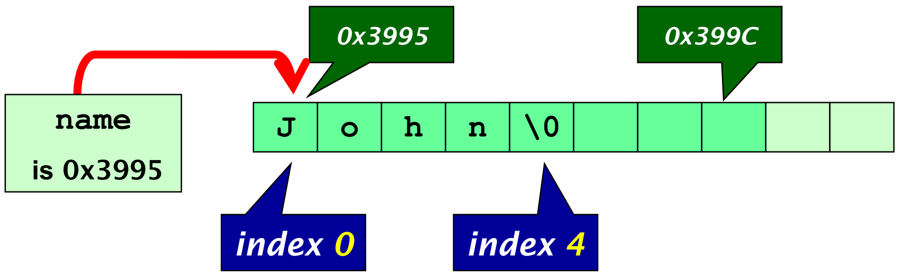
A Char in a String

- The size of a character string is fixed
- Character at position *index*:
  - **`string[index]`**
  - first character has index 0

16

16

A Char in a String (cont.)



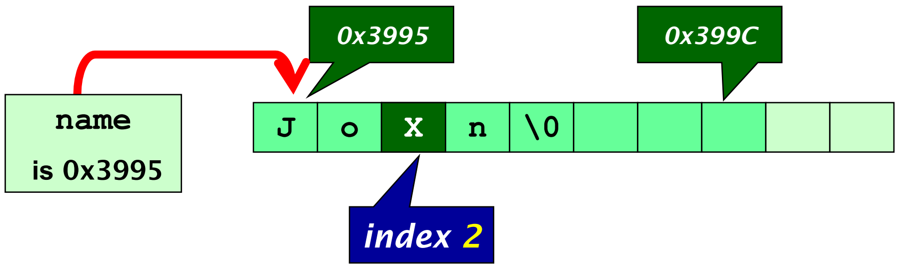
```
char name[8] = "John";
int  i = 2;

printf("Char at index %d is %c.\n", i, name[i]);
```

```
output: Char at index 2 is h.
```

17

17

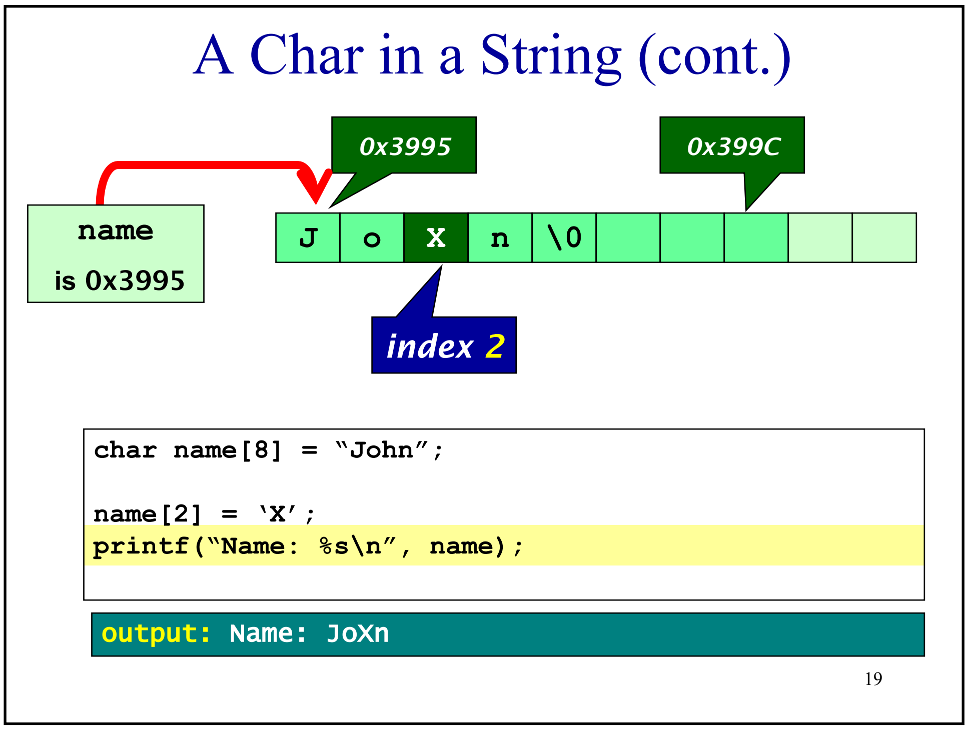
A Char in a String (cont.)



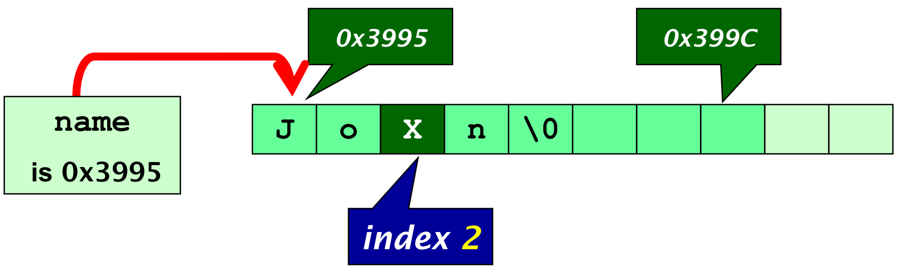
```
char name[8] = "John";

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

18

18



A Char in a String (cont.)

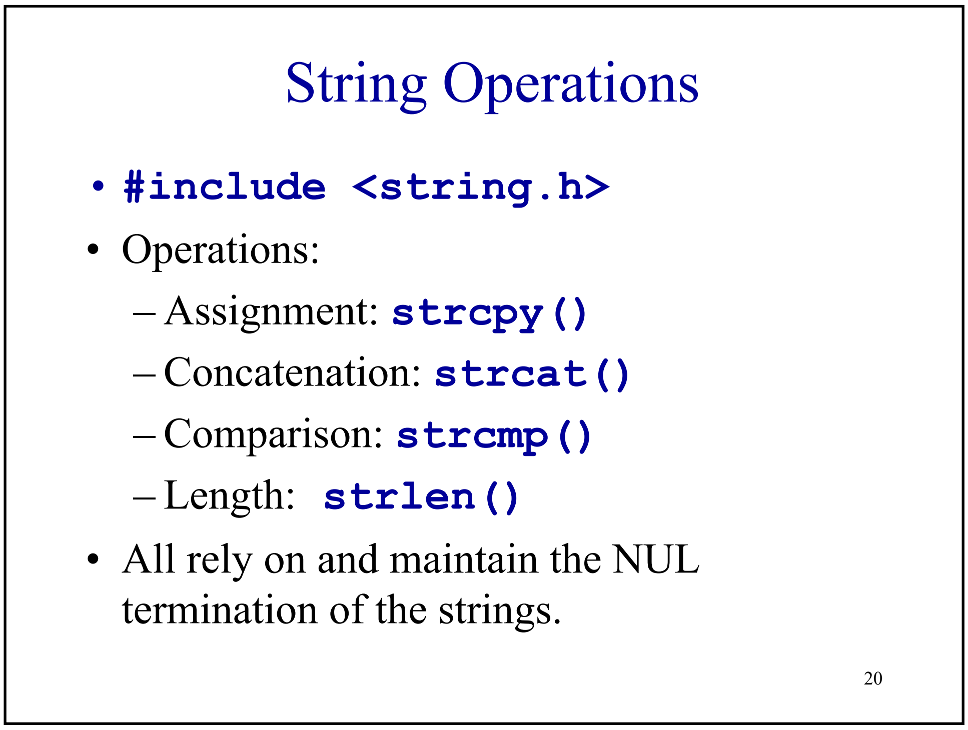


```
char name[8] = "John";

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

```
output: Name: JoXn
```

19

19



String Operations

- **#include <string.h>**
- Operations:
  - Assignment: **strcpy()**
  - Concatenation: **strcat()**
  - Comparison: **strcmp()**
  - Length: **strlen()**
- All rely on and maintain the NUL termination of the strings.

20

20

String Operation: Assignment

```
#include <stdio.h>
#include <string.h>

#define MAXLENGTH 100

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

   strcpy(string1, "Hello World!");
   strcpy(string2, string1);

   return 0;
}
```

```
string1: <garbage>
string2: <garbage>
```

21

21

String Operation: Assignment (cont.)

```
#include <stdio.h>
#include <string.h>

#define MAXLENGTH 100

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

   strcpy(string1, "Hello World!");
   strcpy(string2, string1);

   return 0;
}
```

```
string1: "Hello World!"
string2: <garbage>
```

22

22

String Operation: Assignment (cont.)

```
#include <stdio.h>
#include <string.h>

#define MAXLENGTH 100

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

   strcpy(string1, "Hello World!");
   strcpy(string2, string1);

   return 0;
}
```

```
string1: "Hello World!"
string2: "Hello World!"
```

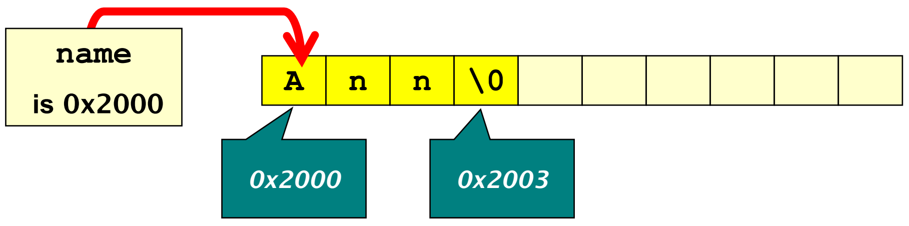
23

23

Common Mistake:

Not enough space

```
char  name[] = "Ann";

strcpy(name, "David");
```



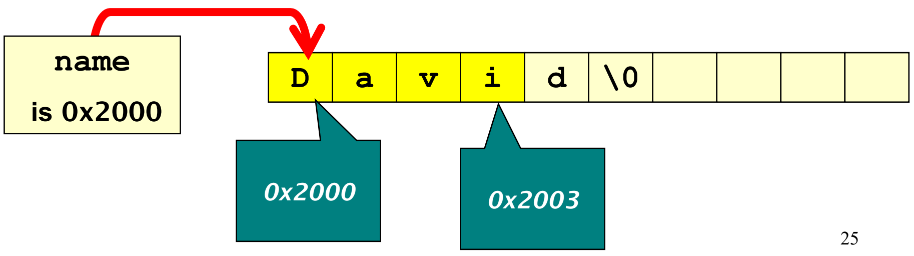
24

24

Common Mistake:

Not enough space

```
char  name[] = "Ann";
strcpy(name, "David");
```



25

25

Caution 1:

Pointer Assignment

Example:

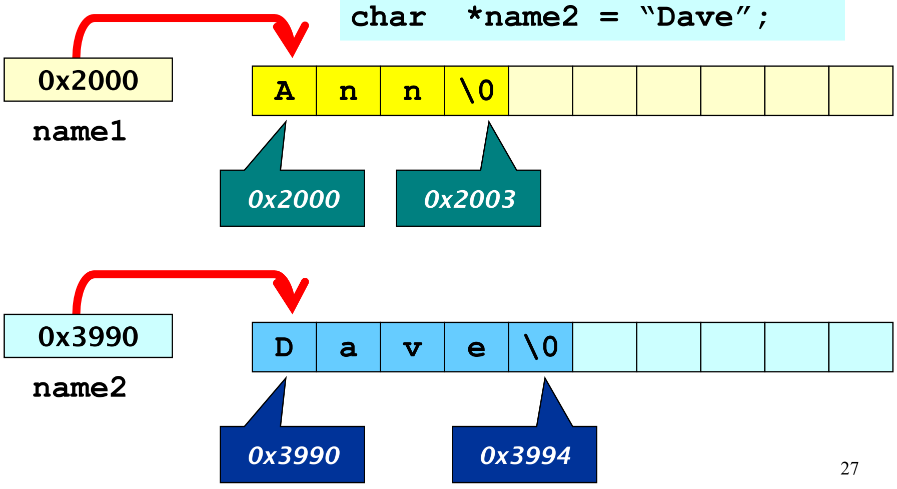
```
char  *name1 = "Ann";
char  *name2 = "Dave";

name2 = name1;
```

26

26

Caution 1:
Pointer Assignment

```
char  *name1 = "Ann";
char  *name2 = "Dave";
```

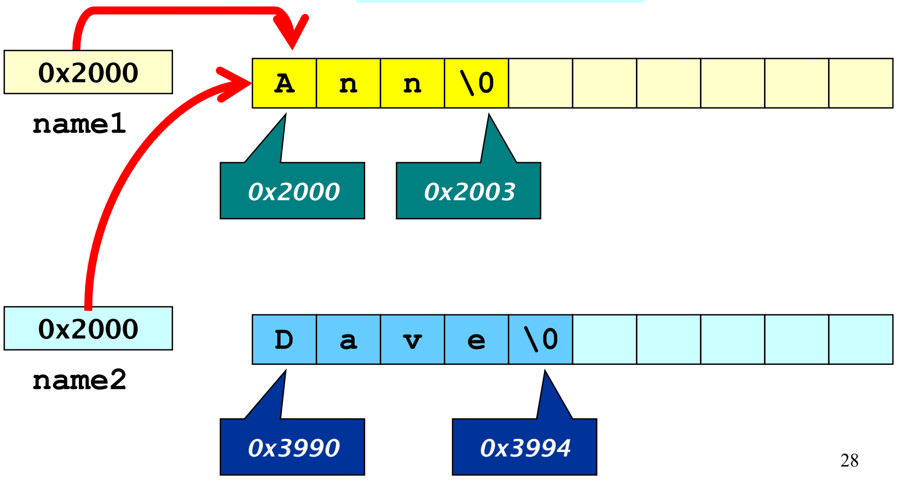


27

27

Caution 1:
Pointer Assignment

```
name2 = name1;
```



28

28

Example: strassign.c

```
#include <stdio.h>
#include <string.h>
#define MAXLENGTH 5
int main()
{
   char name1[MAXLENGTH] = "Ann";
   char name2[] = "Ann";
   char *name3 = "John";
   char name4[MAXLENGTH];

   printf("\nBEFORE\nname1=%s, name2=%s, name3=%s",
          name1, name2, name3);
   strcpy(name1, "Fred");
   strcpy(name2, "Ben");
   strcpy(name4,name1);
   name3 = name2;
   printf("\nAFTER\nname1=%s, name2=%s, name3=%s, name4=%s",
          name1, name2, name3, name4);

   strcpy(name1, "Jack");
   strcpy(name2,"Jim");
   printf("\nLAST\nname1=%s, name2=%s, name3=%s, name4=%s",
          name1, name2, name3, name4);
   return 0;
}
```

String Operation: Concatenation

```
char string1[MAXLENGTH];
char string2[MAXLENGTH];

strcpy(string1, "Hello");
strcpy(string2, ", Good ");

strcat(string1, string2);
strcat(string1, string2);
strcat(string1, "Day!");
```

```
string1: "Hello"
string2: ", Good "
```

String Operation: Concatenation (cont.)

```
char string1[MAXLENGTH];
char string2[MAXLENGTH];

strcpy(string1, "Hello");
strcpy(string2, ", Good ");

strcat(string1, string2);
strcat(string1, string2);
strcat(string1, "Day!");
```

```
string1: "Hello, Good "
string2: ", Good "
```

31

31

String Operation: Concatenation (cont.)

```
char string1[MAXLENGTH];
char string2[MAXLENGTH];

strcpy(string1, "Hello");
strcpy(string2, ", Good ");

strcat(string1, string2);
strcat(string1, string2);
strcat(string1, "Day!");
```

```
string1: "Hello, Good , Good "
string2: ", Good "
```

32

32

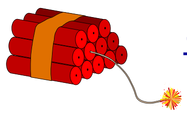
String Operation: Concatenation (cont.)

```
char string1[MAXLENGTH];
char string2[MAXLENGTH];

strcpy(string1, "Hello");
strcpy(string2, ", Good ");

strcat(string1, string2);
strcat(string1, string2);
strcat(string1, "Day!");
```

```
string1: "Hello, Good , Good Day!"
string2: ", Good "
```

33

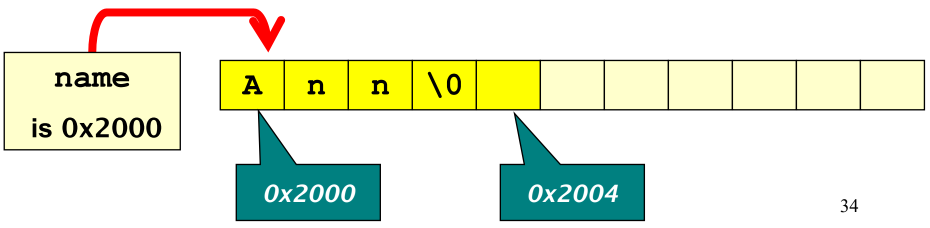
33



Common Mistake:

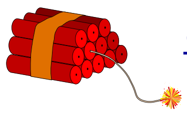
Not enough space

```
char  name[5];

strcpy(name, "Ann");
strcat(name, " Smith");
```



34

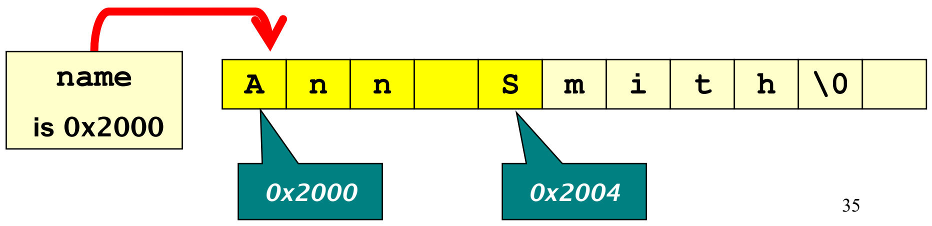
34



Common Mistake:

Not enough space

```
char  name[5];

strcpy(name, "Ann");
strcat(name, " Smith");
```



String Operation: Comparison

```
strcpy(string1, "Apple");
strcpy(string2, "Wax");

if (strcmp(string1, string2) < 0)
{
  printf("%s %s\n", string1, string2);
}
else
{
  printf("%s %s\n", string2, string1);
}
```

- `strcmp()` compares two strings, character by character
- Returns 0 if both strings are identical
- Returns negative integer if the ASCII value of first unmatched character is less than the second
- Returns positive integer if the ASCII value of first unmatched character is greater than the second

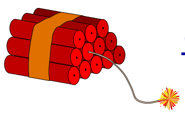
String Operation: Comparison (cont)

```
strcpy(string1, "Apple");
strcpy(string2, "Wax");

if (strcmp(string1, string2) < 0)
{
  printf("%s %s\n", string1, string2);
}
else
{
  printf("%s %s\n", string2, string1);
}
```

```
output: Apple Wax
```

37

37



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);
}
```

38

38

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.

39

39

String Operation: Length

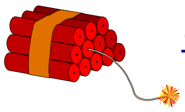
```
char string1[100];

strcpy(string1, "Apple");

printf("%d\n", strlen(string1));
```

```
output: 5
```

Number of char-s before the `\0'

40

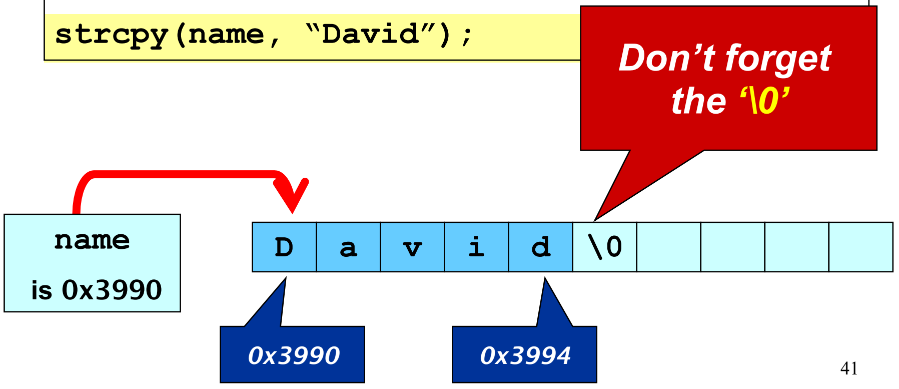
40



Common Mistake:

Not enough space

```
char  name[5];
strcpy(name, "David");
```



41

41

Character Strings as Function Parameters

- Strings as formal parameters are declared as **`char*`** or **`char[]`**
  - *Examples:*
    ```
    void Greet ( char* name )
    void Greet ( char name[] )
    ```
- They point to the first element of the string (array of chars)
- Changes to the string inside the function affect the actual string

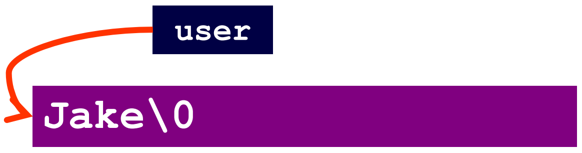
42

42

Example: hello3.c

```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;
}
```



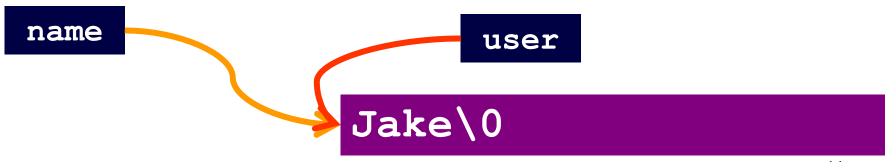
43

43

Example: hello3.c (cont)

```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;
}
```



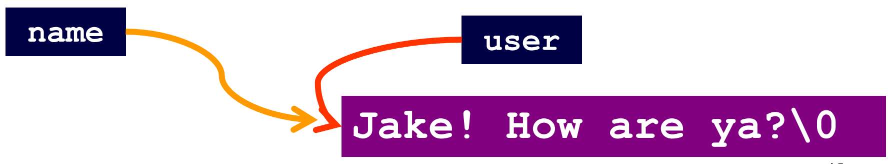
44

44

Example: hello3.c (cont)

```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;
}
```



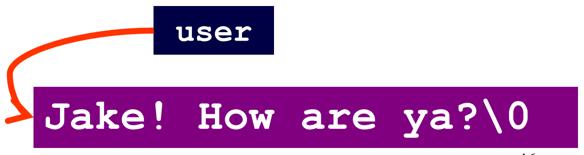
45

45

Example: hello3.c (cont)

```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?");
}
```

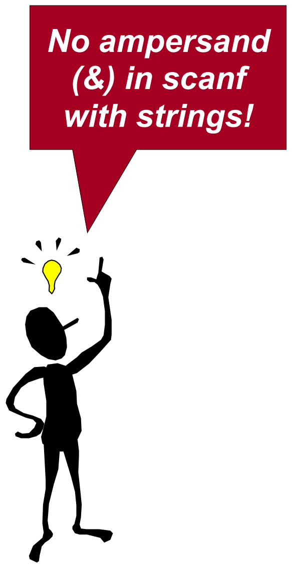
```c
int main()
{
  char user[NAMELEN];

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

  return 0;
}
```



46

46

More of *scanf* demystified



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

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

  return 0;
}
```

47

47

Character Testing Functions

- There is a number of functions defined in `<ctype.h>` that are useful for testing characters
- All the functions takes an `int` as the input parameter, whose value must be representable as an unsigned char
- All the functions return non-zero (true) if the input parameter satisfies the condition described in the function, and zero (false) if not

Function	Description
`int isalnum(int c)`	Checks whether `c` is alphanumeric
`int isalpha(int c)`	Checks whether `c` is alphabetic
`int isdigit(int c)`	Checks whether `c` is a decimal digit
`int islower(int c)`	Checks whether `c` is a lowercase character
`int isupper(int c)`	Checks whether `c` is an uppercase character
`int isspace(int c)`	Checks whether `c` is white-space

48

48

Example of Character Testing

```
#include <stdio.h>
#include <ctype.h>

int main()
{
   char text[] = "a9b7c";
   int i = 0;

   while(text[i] != '\0')
   {
        if (isalpha(text[i])
                printf("%c is an alphabet\n", text[i]);
        else
                printf("%c is not an alphabet\n", text[i]);

        i++;
   }

   return 0;
}
```

Check for the end of string

49

49

Character Conversion Functions

- Two conversion functions that accepts an `int` and returns an `int`

Function	Description
`int tolower(int c)`	Converts uppercase letters to lowercase
`int toupper(int c)`	Converts lowercase letters to uppercase

```
char smallA = 'a';
char bigT = 'T';

char bigA = toupper(smallA);
char smallT = tolower(bigT);
```

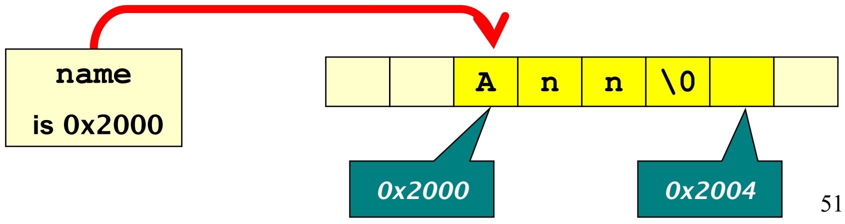
50

50

Arrays of Strings

- We have seen that a string is an array of characters
- The string identifier is the address of the first char in the string, i.e. it is a pointer to the string

```
char  name[5] = "Ann";
```



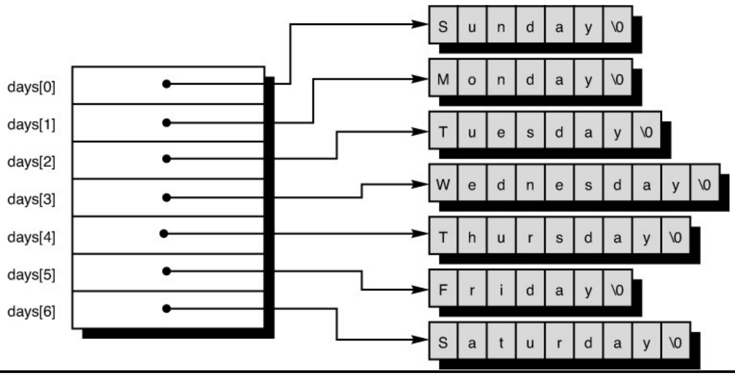
51

51

Arrays of Strings

- An array of strings
  - is an array that contains pointers to strings

```
char *days[7] = {"Sunday", "Monday", "Tuesday",
"Wednesday", "Thursday", "Friday", "Saturday"};
```



52

52

Array of Strings Example 1

```
#include <stdio.h>

int main()
{
   char *days[7] = { "Sunday", "Monday", "Tuesday", "Wednesday",
   "Thursday", "Friday", "Saturday" };
   for (int i=0; i<7; i++)
       printf("%u \t %s\n", days[i], days[i]);

   days[0] = "Sun";
   days[1] = "Mon";
   days[2] = "Tues";
   days[3] = "Wed";
   days[4] = "Thurs";
   days[5] = "Fri";
   days[6] = "Sat";
   for (int i=0; i<7; i++)
       printf("%u \t %s\n", days[i], days[i]);
   return 0;
}
```

Print the string address and the actual string

Changing the pointers to point to other strings

53

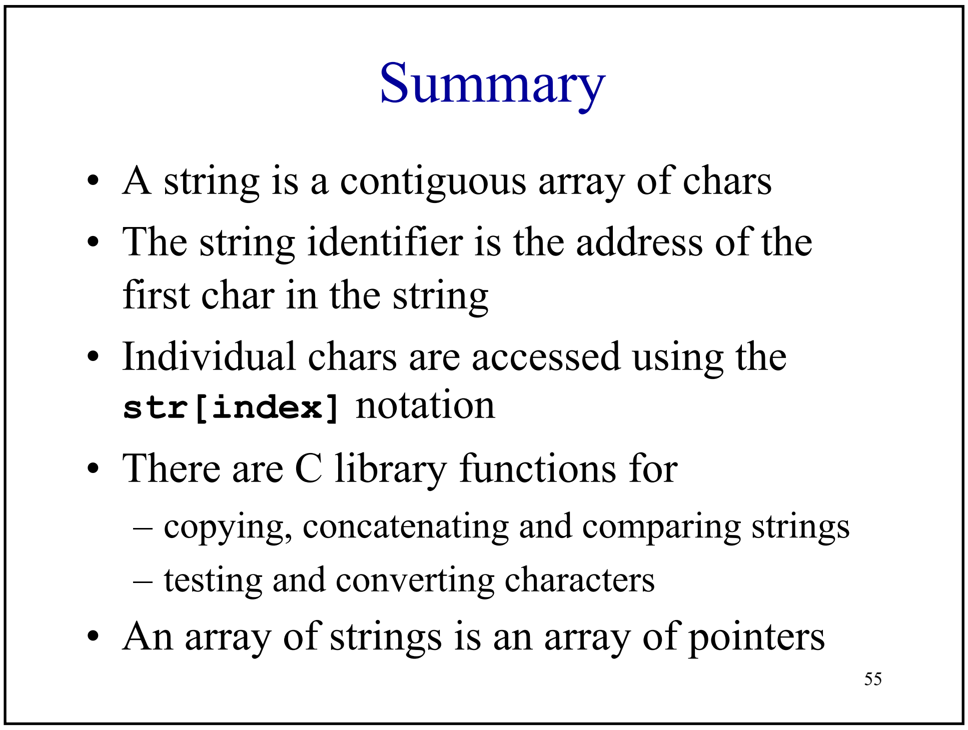
53

Array of Strings Example 2

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main() {
   char *words[5] = {NULL};   // an array of 5 pointers to char
   char temp[100];     // temp storage for a word to be read in

   for (int i=0; i<5; i++) {
        scanf("%s", temp);
        // need to allocate memory to store each string (or word)
        words[i] = calloc(strlen(temp)+1, sizeof(char));
        if (words[i] == NULL) {
                printf("Calloc failed to allocate memory\n");
                return 1;
        }
        strcpy(words[i], temp);        // copy string
   }

   for (int i=0; i<5; i++) {
        printf("%s\n", words[i]);
        free(words[i]);
   }
   return 0;
}
```

54

54



Summary

- A string is a contiguous array of chars
- The string identifier is the address of the first char in the string
- Individual chars are accessed using the **`str[index]`** notation
- There are C library functions for
  - copying, concatenating and comparing strings
  - testing and converting characters
- An array of strings is an array of pointers

55

55