Fall 2006 Handout 4

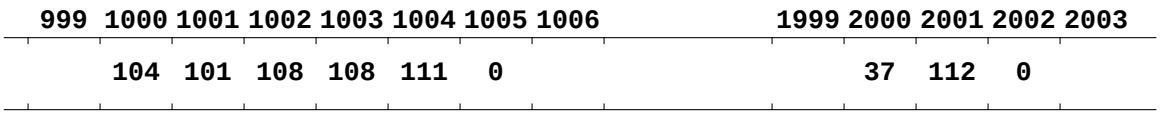
Double quotes: K&R pp. 30, 38−39, 194; King pp. 14, 242

We've seen a double quoted string in only one place so far: as the first argument of **printf**. Unlike single quotes, double quotes can enclose any number of characters (or even none at all):

```
1 printf("hello");
2 printf("Once I built a railroad,\n");
3 printf("");                          /* useless statement */
```

Characters enclosed in double quotes are always loaded into consecutive bytes of memory before the program begins to run. Then the double quotes always load one extra byte containing the number 0 after the last character. In the following example, 104 is the ASCII code of lowercase **h**, 101 is the ASCII code of lowercase **e**, 37 is the ASCII code of **%**, and 112 is the ASCII code of lowercase **s**.

```
4 printf("%p\n", "hello");            /* It prints 1000. */
5 printf("%s\n", "hello");            /* It prints hello. */
```



Like every other expression, a double quoted string such as **"hello"** has a value. This value is the address of the first **char** in the string, in this case 1000. Thus deep inside the machine, the above **printf**'s receive the value 1000 as their second argument.

The first argument of **printf** is also the address of a **char** in memory. **printf** loops through the **char**'s of memory starting at this address, printing every **char** until it encounters a **char** containing the number 0. It then stops without printing the 0 **char**. If it encounters along the way two consecutive **char**'s containing **'%'** and **'d'**, etc., **printf** outputs the value of a subsequent argument in place of the **%d**.

The first arguments of **printf** does not necessarily have to be a double quoted string. It could be any expression whose value is the address of the first of a string of **char**'s in memory, ending with a **char** containing the number 0. **printf** doesn't know or care whether you use a double quoted string or another kind of expression—all it receives is a number (1000, in the above example).

It is a convention in C that every string ends with a **char** containing the number 0. Most of the string functions on K&R pp. 249−250, King pp. 529−536 take as an argument the address of the first character of a string, and process the characters from that point onward until they encounter a **char** containing the number 0.

A double quoted string that's too long to fit on one line:
K&R pp. 38, 194, 260 bullet 3; King pp. 240−241

The following three statements all produce the same output. All three store the same 36 **char**'s (including the **\n** and the terminating **'\0'**) into consecutive memory addresses before the program starts to run.

```
1     printf("supercalifragilisticexpialidocious\n");
2
```

```
 3      printf("supercalifragilistic" "expialidocious\n");
 4
 5      printf("supercalifragilistic"
 6          "expialidocious\n");
```

Store a string in an array of characters and print it: K&R pp. 30, 154; King pp. 241−242

Every variable in C holds one number. To hold a string of characters, use an array of **`char`**'s. Each **`char`** will hold the ASCII code number of one character.

It looks like the **`printf`**'s in lines 16−17 are receiving an array of **`char`**'s as their second argument. But the value of the expression **`s`** is actually a number: it's the address of **`s[0]`**.

The **`%p`** in line 16 tells **`printf`** to print the number itself. The **`%s`** in line 17 tells **`printf`** to print every **`char`** in memory starting at that address until a **`char`** containing 0 is encountered. The **`char`** containing 0 is not printed.

```
 1 #include <stdio.h>
 2
 3 main()
 4 {
 5      char s[6];
 6      int i;
 7      char *p;
 8
 9      s[0] = 'h';
10      s[1] = 'e';
11      s[2] = 'l';
12      s[3] = 'l';
13      s[4] = 'o';
14      s[5] = '\0';
15
16      printf("The string starts at address %p.\n", s);
17      printf("The string is %s.\n", s);
18      printf("The string is %s with the first character removed.\n", s + 1);
19
20      /* Print one character at a time. */
21      for (i = 0; s[i] != '\0'; ++i) {
22          printf("%c", s[i]);              /* or putchar(s[i]); */
23      }
24      printf("\n");                        /* or putchar('\n'); */
25
26      /* Print one character at a time. */
27      for (p = s; *p != '\0'; ++p) {
28          printf("%c", *p);                /* or putchar(*p); */
29      }
30      printf("\n");                        /* or putchar('\n'); */
31 }
```

```
The string starts at address 1000.          may be different on your machine
The string is hello.
The string is ello with the first character removed.
hello
hello
```

Initialize a string: K&R p. 86; King pp. 243−245

Instead of executing lines 9−14 above, we could have created the array with an initial value in each element. The following three declarations all do the same thing. The first two are for pedagogical purposes; please write only the third. In all three, the **6** can and should be omitted.

```
1 char s[6] = {104, 101, 108, 108, 111, 0};       /* like the array of months */
2 char s[6] = {'h', 'e', 'l', 'l', 'o', '\0'};
3 char s[6] = "hello";
```

Three ways to printf a string

If you're going to print a string only once, write the string in double quotes as the first argument of **printf** (line 8). But if you're going to print the same string with several different **printf**'s, write the string once and for all in an array (line 5) and let the **printf**'s print the array (line 11).

```
 1 #include <stdio.h>
 2
 3 main()
 4 {
 5     char s[] = "hello\n";
 6     char t[] = "I agree 100%.\n";
 7
 8     printf("hello\n");
 9
10     printf(s);          /* Okay, but this is a risky way to print s string */
11     printf("%s", s);
12
13     printf(t);          /* bug */
14     printf("%s", t);
15 }
```

Use subscripts to draw a little rocket in a char array ==>

If the carriage return **\r** (K&R p. 38; King p. 119) doesn't work, change it to **\033[;H\033[2J** (clear screen, home cursor). If the **sleep** is too long, remove it.

—On the Web at
http://i5.nyu.edu/~mm64/x52.9232/src/rocket1.c

```
 1 #include <stdio.h>
 2 #include <unistd.h> /* for sleep; may be different or unnecessary on your machine */
 3 #define  N 80         /* length of line */
 4
 5 main()
 6 {
 7     char line[N];        /* background, all blanks */
 8     int i;
 9
10     /* Initialize the array to all blanks, terminated with a '\0'. */
11     for (i = 0; i < N - 1; ++i) {
12         line[i] = ' ';
13     }
14     line[i] = '\0';              /* Put '\0' into line[N-1]. */
15
16     for (i = 0; i < N - 3; ++i) {
17
18         /* Erase the last char of the previous rocket, if there was one. */
```

```
19          if (i > 0) {
20              line[i-1] = ' ';
21          }
22
23          /* Draw the rocket. */
24          line[i]   = '=';
25          line[i+1] = '=';
26          line[i+2] = '>';
27
28          printf("%s\r", line);
29          fflush(stdout);      /* Output the above characters immediately. */
30          sleep(1);            /* for 1 second */
31      }
32  }
```

Use a pointer to draw the rocket in a char array

—On the Web at
http://i5.nyu.edu/~mm64/x52.9232/src/rocket2.c

```
 1  #include <stdio.h>
 2  #include <unistd.h> /* for sleep; may be different or unnecessary on your machine */
 3  #define  N   80          /* length of line */
 4
 5  main()
 6  {
 7      char line[N];            /* background, all blanks */
 8      char *p;
 9
10      /* Initialize the array to all blanks, terminated with a '\0'. */
11      for (p = line; p < line + N - 1; ++p) {
12          *p = ' ';
13      }
14      *p = '\0';                   /* Put '\0' into line[N - 1]. */
15
16      for (p = line; p < line + N - 3; ++p) {
17
18          /* Erase the last char of the previous rocket, if there was one. */
19          if (p > line) {
20              p[-1] = ' ';
21          }
22
23          /* Draw the rocket. */
24          p[0] = '=';
25          p[1] = '=';
26          p[2] = '>';
27
28          printf("%s\r", line);
29          fflush(stdout);
30          sleep(1);
31      }
32  }
```

In lines 24−26 (and 20) above, we pretend that **p** is a portable array of **char**'s starting at the base of the rocket: **p[0]**, **p[1]**, **p[2]**. You can use negative subscripts, too, as long as you stay within the **line**

array.

You can treat any **char** in memory as the start of an array of **char**'s by aiming a ''pointer to **char**'' at it. Similarly, you can treat any **int** in memory as the start of an array of **int**'s, etc. This allows you to pretend that you have a portable array starting wherever you're currently at work. Compare the two versions of lines 24−26:

```
line[i]   = '=';                    p[0] = '=';
line[i+1] = '=';                    p[1] = '=';
line[i+2] = '>';                    p[2] = '>';
```

▼ Homework 4.1: the Allied Chemical Building

Make an infinite **for** loop to simulate the marquee in Times Square which rotates from right to left. Write your own headline by declaring an initialized array. Use all uppercase, with a period between words and three periods at the end of the message:

```
char message[] = "CUOMO.SAYS.HE.NOW.WISHES.HE.HAD.RUN...";
```

At the start of each loop, print the entire **message** with a **printf %s**. (Do not print each character individually with a **printf %c**.) Then copy the first character of the **message** into a **char** variable named **save**. Write an inner **for** loop to move the remaining characters in **message** one byte to the left. The induction variable of this loop will be a

```
char *p;
```

Stop the inner **for** loop when you reach the **'\0'** at the end of the array. Finally, store the first character at the end of the **message** right before the **'\0'**. Be sure that the **'\0'** remains untouched in the last byte of **message**: there is no need to copy or move it. **sleep** for a second if the display is too fast to read.

You get no credit if you make any of the following mistakes. This program accepts no input—do not write **scanf** or **getchar**. Do not create the infinite loop by any way other than the **for (;;)** in Handout 1. There must be exactly three variables: the array **message**, the **char save**, and the pointer **p** (plus one more variable to be the induction variable in an extra **for** loop if you have no **sleep** function and you wish to slow down). There must be exactly one **printf**, two **for**'s (or three **for**'s if you wish to slow down), and no **while** or **if**. **fflush** if necessary to get smooth motion. Do not write any number larger than 3 in your program—i.e., 38 or 39. Do not use **sizeof**.

Do not overwrite the bytes of memory adjacent to the start and end of the **message**—you might damage other variables. The **printf** will print the same number of characters each time it is executed. This number must be equal to the number of characters you enclosed in double quotes when initializing the **message** (not counting the terminating **'\0'**). Do not display the same character twice with one **printf**, as is the first **O** in this **CUOMO**:

```
OMO.SAYS.HE.WISHES.NOW.HE.HAD.RUN...CUO
```

▲

Input a line one character at a time: K&R p. 29−30; King pp. 248−249

Unless you undertake special negotiations with your operating system, your program has no access to the characters that the user types until he or she presses **RETURN**. Then the entire line becomes available to **getchar** and **scanf**.

The

```
while ((c = getchar()) != EOF) {
```

examples in K&R pp. 15−24; show how to input a string one character at a time. These characters may be stored in an array:

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



```
 2 #define N   256     /* maximum string length, including the '\0' */
 3
 4 main()
 5 {
 6     char line[N];
 7     int i;
 8     int c;
 9
10     printf("Please type a line and press RETURN.\n");
11
12     for (i = 0; (c = getchar()) != '\n'; ++i) {
13         line[i] = c;
14     }
15     line[i] = '\0';
16
17     printf("The line was %s.\n", line);
18 }
```

There are actually three conditions, any one of which should stop the loop:

```
12     for (i = 0; i < N - 1 && (c = getchar()) != EOF && c != '\n'; ++i) {
```

```
19 #include <stdio.h>
20 #define N   256     /* maximum line length, including the '\0' */
21
22 main()
23 {
24     char line[N];
25     char *p
26     int c;
27
28     printf("Please type a line and press RETURN.\n");
29
30     for (p = line; p < line+N-1 && (c = getchar()) != EOF && c != '\n'; ++p) {
31         *p = c;
32     }
33     *p = '\0';
34
35     printf("The line was %s.\n", line);
36 }
```

Input a line with fgets: K&R pp. 164, 247

```
 1 #include <stdio.h>
 2 #define N   256     /* maximum line length, including the '\0' */
 3
 4 main()
 5 {
 6     char line[N];
 7
 8     printf("Please type a line and press RETURN.\n");
 9     fgets(N, line, stdin);
10     line[strlen(line) - 1] = '\0';              /* Zap the newline. */
11     printf("The line was %s.\n", line);
12 }
```

Change lines 9−11 to examine the return value of **fgets**:

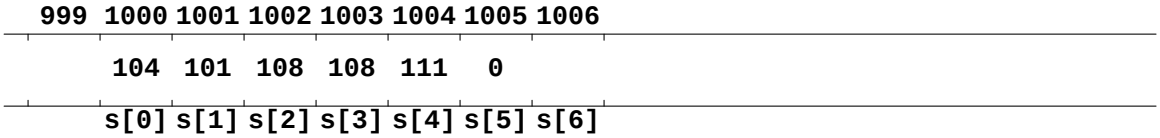
```
 9      if (fgets(N, line, stdin) == line) {
10          line[strlen(line) - 1] = '\0';/* zap the newline */
11          printf("The line was %s.\n", line);
12      } else {
13          printf("End-of-file or other error encountered.\n");
14      }
```

Input a word with %s: K&R pp. 158−159; King pp. 247−248

Give **scanf %s** the memory address at which to store a string, for example the address of an array of **char**'s. **scanf %s** will ignore leading white space in the user's input, and will then store the non-white input characters into consecutive bytes of memory starting at the address you specified. **scanf %s** will stop when it encounters a white space input character (e.g., the newline that the user input by pressing **RETURN**); this character is not stored in memory with the others. Instead, **scanf %s** deposits a terminating **'\0'** in memory after the last character, so you can **printf** this string later.

```
 1 #include <stdio.h>
 2 #define N   256     /* maximum word length, including the '\0' */
 3
 4 main()
 5 {
 6     char word[N];
 7
 8     printf("Please type a word and press RETURN.\n");
 9     scanf("%s", word);
10     printf("The word was %s.\n", word);
11 }
```

```
Please type a word and press RETURN.
helloRETURN
The word was hello.
```



Warning: **scanf %s** does not know the length of the array of **char**'s into which it is writing. A malicious user can therefore make **scanf %s** write beyond the end of the array.

Input several words with one scanf

```
 1 #include <stdio.h>
 2 #define N   256     /* maximum string length, including the '\0' */
 3
 4 main()
 5 {
 6     char first[N];
 7     char middle[N];
 8     char last[N];
 9
10     printf("Please type your full name and press RETURN.\n);
11     printf("Type one or more blanks between your first and middle names,\");
12     printf("and one or more blanks between your middle and last names,\n");
```



```
13     printf("but no blanks before you press RETURN.  For example,\n");
14     printf("Charles Foster KaneRETURN\n");
15
16     scanf("%s", first);
17     scanf("%s", middle);
18     scanf("%s", last);
19
20     printf("%s %s %s\n", first, middle, last);
21 }
```

You can combine lines 16−18 into

```
22 scanf("%s%s%s", first, middle, last);     /* K&R p. 159 */
```

Other formats for string input: K&R pp. 245−246; King pp. 492−496

Suppose the user types

```
Meretzky, Mark:134-46-0566
```

as input in the two programs below. Then each **`scanf`** in the first program would deposit the specified number of input characters into the array, followed by one character containing **`'\0'`**.

A dash within square brackets counts as an abbreviation only when it is not adjacent to either of the square brackets. Type no space on either side of the dash. The character to the left of the dash must have a smaller ASCII code than the character to the right of the dash.

```
 1 #include <stdio.h>
 2 #define  N   256     /* maximum string length, including the '\0' */
 3
 4 main()
 5 {
 6     char s[N];
 7     printf("Please type a line and press RETURN.\n");
 8
 9     scanf("%s", s);              /* Input 9 characters. */
10     scanf("%[ABCDEFGHIJKLMNOPQRSTUVWXYZ]", s);        /* Input 1 character. */
11     scanf("%[A-Z]", s);          /* Input 1 character. */
12     scanf("%[A-Za-z]", s);       /* Input 8 characters. */
13     scanf("%[^:]", s);           /* Input 14 characters. */
14     scanf("%[^0-9]", s);         /* Input 15 characters. */
15     scanf("%[^\n]", s);          /* Input 26 characters. */
16     scanf("%4s", s);             /* Input 4 characters. */
17 }
```

You can use these formats to split up a line of input:

```
 1 #include <stdio.h>
 2 #define  N   256     /* maximum string length, including the '\0' */
 3
 4 main()
 5 {
 6     char last[N];
 7     char first[N];
 8     char ss[N];          /* social security number */
 9     char dummy[N];       /* See * on K&R pp. 157, 245; King pp. 496, 512
10                         to eliminate need for dummy. */
11
```

```
12      printf("Please type a line and press RETURN.\n");
13
14      scanf("%[^A-Za-z]", dummy);     /* Skip leading white space, if any. */
15      scanf("%[A-Za-z]", last);
16      scanf("%[^A-Za-z]", dummy);     /* Skip the comma and blank. */
17      scanf("%[A-Za-z]", first);
18      scanf("%[^0-9]", dummy);        /* Skip the colon. */
19      scanf("%[0-9-]", ss);
20
21      printf("%s %s %s\n", first, last, ss);
22  }
```

The difference between strlen and sizeof: K&R pp. 249−250; King pp. 254−255

The **strlen** function counts the bytes in memory, starting at the address you give it, until it encounters a **'\0'**. The **'\0'** is not included in the count. **strlen** doesn't know which array it is examining, or even whether it is examining an array. All it knows is to direct its attention to the address you give it.

The **sizeof** unary operator does not examine the contents of memory at all; instead, it looks at the declaration that created the array. The **strlen** depends on the values of the bytes in memory, and hence can change; the **sizeof** an array never changes.

```
 1  #include <stdio.h>
 2  #include <string.h>
 3
 4  main()
 5  {
 6      char s[] = "hello";      /* Create an array of 6 char's. */
 7      int i;
 8      char *p;
 9
10      for (i = 0; s[i] != '\0'; ++i) {
11      }
12      printf("The length is %d.\n", i);
13
14      for (p = s; *p != '\0'; ++p) {
15      }
16      printf("The length is %d.\n", p - s);
17
18      printf("The strlen is %d.\n, strlen(s));
19      printf("The sizeof s is %d.\n\n", sizeof s);
20
21      s[3] = '\0';         /* Clobber the second 'l' in hello. */
22      printf("The strlen is now %d.\n, strlen(s));
23      printf("sizeof s is still %d.\n", sizeof s);
24  }
```

```
The length is 5.
The length is 5.
The strlen is 5.
The sizeof s is 6.

The strlen is now 3.
The sizeof s is still 6.
```

▼ Homework 4.2: Print a number with commas: 12,345,678

Change the following program to use two pointers instead of two integer indices. Name the pointers **s** and **d**.

Combine lines 24−26 into a single statement, and combine lines 28−29 into a single statement. see the eye-popping ***s++ = *t++** atop K&R p. 106

```
1 /* before */                          /* after */
2 c = *s;                              c = *s++;
3 s++;

4 *d = c;                              *d++ = c;
5 d++;

6 *d = *s;                             *d++ = *s++;
7 s++;
8 d++;
```

The diagram shows each byte in the arrays for the input string **12345678**. Subscript numbers are written in brackets. For example, **source[0]** contains **'1'**, which is really the byte **00110001**, and **source[8]** contains **'\0'**, which is really the byte **00000000**. Empty boxes contain undefined values, i.e., garbage.

source

'1'	'2'	'3'	'4'	'5'	'6'	'7'	'8'	'\0'						
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]

dest

'1'	'2'	','	'3'	'4'	'5'	','	'6'	'7'	'8'	'\0'				
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]

—On the Web at
http://i5.nyu.edu/~mm64/x52.9232/src/comma.c

```
 1 /* Input an integer and output it with commas every three digits. */
 2
 3 #include <stdio.h>
 4 #include <string.h>
 5
 6 #define  N   256     /* maximum string length, including the '\0' */
 7
 8 main()
 9 {
10     char source[N];      /* hold the digits without commas */
11     char dest[N];        /* hold the digits with commas */
12
13     int s = 0;       /* index of each char in source */
14     int d = 0;       /* index of each char in dest */
15     int count;       /* count groups of three digits */
16
17     printf("Please input an integer and press RETURN.\n");
18     scanf("%s", source);
19
```



```
20      /* Copy the chars from source into dest, including the trailing
21      '\0'.  Insert a comma every three digits. */
22
23      for (count = strlen(source); count >= 0; --count) {
24          dest[d] = source[s];     /* Copy a character. */
25          s++;
26          d++;
27          if (count % 3 == 1 && count > 1 && dest[d-1] != '-') {
28              dest[d] = ',';
29              d++;
30          }
31      }
32
33      printf("%s\n", dest);
34  }
```

Use the following test data.

```
                    0
                    1                                  -1
                   12                                 -12
                  123                                -123
                 1234                               -1234
           2147483647                         -2147483648
```

▲

String functions: K&R pp. 249−250; King pp. 529−536

The **strstr** in line 18 searches for a little string in a big string:

—On the Web at
http://i5.nyu.edu/~mm64/x52.9232/src/sugar.c

```
 1  #include <stdio.h>
 2  #include <stdlib.h>
 3  #include <string.h>
 4  #define N 256
 5
 6  int main()
 7  {
 8      char food[N];
 9
10      printf("What would you like to eat? ");
11      if (fgets(food, N, stdin) != food) {
12          printf("End of input encountered.\n");
13          return EXIT_FAILURE;
14      }
15
16      printf("You can't eat that--");
17
18      if (strstr(food, "sugar") == NULL) {
19          printf("it turns to sugar in your stomach!\n");
20      } else {
21          printf("it has sugar in it!\n");
22      }
23
24      return EXIT_SUCCESS;
```

```
25 }
```

```
What would you like to eat? sugar plums
You can't eat that--it has sugar in it!

What would you like to eat? dumplings
You can't eat that--it turns to sugar in your stomach!
```

The **strstr** in line 18 searches for a little string in a big string:

—On the Web at
http://i5.nyu.edu/~mm64/x52.9232/src/piglat1.c

```
 1 /* Convert one word of English to Pig Latin. */
 2
 3 #include <stdio.h>
 4 #include <string.h>
 5 #define N   256     /* maximum string length, including the '\0' */
 6
 7 main()
 8 {
 9     char english[N];
10     char piglatin[N];
11     char suffix[] = "?ay";
12
13     printf("Please type a word and press RETURN.\n");
14     scanf("%s", english);
15
16     suffix[0] = english[0];
17     strcpy(piglatin, english + 1);
18     strcat(piglatin, suffix);
19
20     printf("In Pig Latin, %s is %s.\n", english, piglatin);
21 }
```

The **strchr** in line 18 searches for a character in a big string:

—On the Web at
http://i5.nyu.edu/~mm64/x52.9232/src/piglat2.c

```
 1 /* Convert one word of English to Pig Latin.  Do not move an initial vowel
 2 to the end.  Preserve capitalization. */
 3
 4 #include <stdio.h>
 5 #include <string.h>
 6 #include <ctype.h>
 7 #define N   256     /* maximum string length, including the '\0' */
 8
 9 main()
10 {
11     char english[N];
12     char piglatin[N];
13     char suffix[] = "?ay";
14
15     printf("Please type a word and press RETURN.\n");
16     scanf("%s", english);
17
18     if (strchr("AEIOUaeiou", english[0]) != NULL) {
```

```
19          suffix[0] = 'w';
20          strcpy(piglatin, english);
21      } else {
22          suffix[0] = english[0];
23          strcpy(piglatin, english + 1);
24          if (isupper(suffix[0])) {
25              suffix[0] = tolower(suffix[0]);
26              piglatin[0] = toupper(piglatin[0]);
27          }
28      }
29      strcat(piglatin, suffix);
30
31      printf("In Pig Latin, %s is %s.\n", english, piglatin);
32  }
```

```
Please type a word and press RETURN.
Hello
In Pig Latin, Hello is Ellohay.
```

```
Please type a word and press RETURN.
apple
In Pig Latin, apple is appleway.
```

▼ Homework 4.3: required reading: string functions (not to be handed in)

Read the descriptions of the string functions in K&R pp. 249−250; King pp. 529−536. Could you write a program to do what each of them does?

▲

□