Numbers and Array

EECS 2031

Song Wang
wangsong@eecs.yorku.ca
eecs.yorku.ca/~wangsong/

Acknowledgement

- Some of the covered materials are based on previous EECS2031 offerings:
  - Uyen Trang (UT) Nguyen, Pooja Vashisth, Hui Wang, Manos Papagelis

Binary number (base 2)

- A binary number is a number that includes only ones and zeroes.
- The number could be of any length
- The following are all examples of binary numbers

```
0       10101
1       0101010
10      1011110101
01      0110101110
111000      000111
```

- Another name for binary is base-2 (pronounced "base two")

Decimal (base 10)

- Uses positional representation
- Each digit corresponds to a power of 10 based on its position in the number
- The powers of 10 increment from 0, 1, 2, etc. as you move right to left

$$1{,}479 = 1 * 10^3 + 4 * 10^2 + 7 * 10^1 + 9 * 10^0$$

Equivalence of Binary and Decimal

- Every Binary number has a corresponding Decimal value (and vice versa)

- Examples:

Binary Number	Decimal Equivalent
1	1
10	2
11	3
...	...
1010111	87

Hexadecimal (base 16)

- A “hexadecimal” number is a number where each digit may be one of sixteen possible values.

- The possible values for a hexadecimal digit are:
  0 1 2 3 4 5 6 7 8 9 A B C D E F

- A digit of
  “A” stands for the number 10
  “B” stands for the number 11
  “C” stands for the number 12
  “D” stands for the number 13
  “E” stands for the number 14
  “F” stands for the number 15

Hexadecimal numbers

- The following are all valid hexadecimal nubmers

  A

  9 (yes, a hexadecimal number does not HAVE TO contain letters)

  1001 (yes, a hexadecimal number does not HAVE TO contain letters)

  9C5

  BFE

- To understand what a specific hexadecimal number means, you can convert it into an equivalent decimal number.

Converting a Hexadecimal number to Decimal

- **The value of hexadecimal A12F is decimal 41,263. See below:**

4096 (i.e 16^3) 256 (i.e 16^2) 16 (i.e 16^1) 1 (i.e 16^0)

A 1 2 F

15 X 1= 15

2 X 16 = 32
1 X 256 = 256

10 X 4096 = 40,960

Answer: 41,263

Another example

0xABC =?

Binary	Decimal	Hexadecimal
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	10	A
1011	11	B
1100	12	C
1101	13	D
1110	14	E
1111	15	F
10000	16	10

Octal Numbers (base 8)

- Like the hexadecimal system, the octal system provides a convenient way to express binary numbers and codes.
- However, it is used less frequently than hexadecimal in conjunction with computers and microprocessors to express binary quantities for input and output purposes.
- The octal system is composed of eight digits, which are:

  0, 1, 2, 3, 4, 5, 6, 7

- Counting in octal is similar to counting in decimal, except that the digits 8 and 9 are not used.

Integer Constants

- Integer constants can be expressed in three different ways

1. **Decimal** [base 10]
   - `int x = 31`

same in Java

2. **Octal** [base 8]
   - Start with zero 0
   - `int x = 037` (31 in decimal)

same in Java

3. **Hexadecimal** [base 16]
   - Start with `0x` or `0X`
   - `int x = 0x1F` (31 in decimal)

same in Java

```c
/* salute the world */

main ()
{
  int x = 31;
  int x2 = 037;
  int x3 = 0x1F;

  printf( "%d\n", x );
  printf( "%d\n", x2 );
  printf( "%d\n", x3 );

}
```

same in Java

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
ea06 313 % gcc binaryLiteral0.c
ea06 314 % a.out
31
31
31
ea06 315 %
```

Binary literal is not C standard, although it might work

```
4   /* salute the world */
5
6   main ()
7   {
8     int x = 30;
9     int x2 = 037;
10    int x3 = 0x1F;
11    int x4 = 0b01111011;
12    printf( "Hello, world\n" );
13  }
14
15
```

okay in Java

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
ea57 318 % gcc -pedantic-errors binaryLiteral.c
binaryLiteral.c: In function ‘main’:
binaryLiteral.c:11:12: error: binary constants are a GCC extension
   int x4 = 0b01111011;
            ^
ea57 319 %
```

Integer Constants (cont.)

- We can specify type qualifier at the end:
  - 'u' or 'U' ⇨ **unsigned (int)**
  - 'l' or 'L' ⇨ **long (int)**
  - nothing ⇨ **int**

same in Java

- E.g.

`5`	as an	**"(signed) (decimal) int" 5**
`5U`	as an	**"unsigned (decimal) int" 5**
`5L`	as a	**"(signed) long (int)" 5**
`5UL or 5ul`	as an	**"unsigned long (int)" 5**
`037`	as an	**"(signed) int (oct)" decimal: 31**
`0x32dUL`	as an	**"unsigned long (int) in hex 32d"**
`059`	as an	?
`0x39G2`	as an	?

Floating Point Constants

- All floating point constants contain a decimal point('.') and/or an exponent ('e' or "E")
  - E.g. **`1.532  3e5  4.112e-10`**
  - **`5.3e12`** == 5.3×10^{12}
  - **`printf("%E %e", 0.00137, 123.025);`**

    1.370000E-03 1.230250e+02

- Floating point constants are of type 'double'
  - Nothing – means **"double"** e.g., **`double x = 1.532`**

    same in Java

  - 'f' or 'F' - means **"float"** e.g. **`float x = 1.532f`**
    `float x = 1.532` OK

    same in Java

  - 'l' or 'L' - means **"long double"** e.g. **`long double x=1.5L`**

    same in Java

Arrays

int a [10];

double x [20][30];

- Sequence of values **with the same type**
- Accessed using an index:
  - Example: a[5]
  - Indices between 0 and size-1
- Initialization
  - local variable: random data
  - global/**static** variables: **“0”**

```c
#include<stdio.h>

int foo();

int main(){
   int a [2];
   static int b[2];
   int c [2] ={0};

   printf("a[1] is: %d\n", a[1]);
   printf("b[1] is: %d\n", b[1]);
   printf("c[1] is: %d\n", c[1]);
   foo();
}

int foo(){
   int a[2];
   static int b[2];
   int c[2] = {0};

   printf("Inside foo \n");
   printf("a[1] is: %d\n", a[1]);
   printf("b[1] is: %d\n", b[1]);
   printf("c[1] is: %d\n", c[1]);
}
```

```c
#include<stdio.h>

int foo();

int main(){
   int a [2];
   static int b[2];
   int c [2] ={0};

   printf("a[1] is: %d\n", a[1]);
   printf("b[1] is: %d\n", b[1]);
   printf("c[1] is: %d\n", c[1]);
   foo();
}

int foo(){
   int a[2];
   static int b[2];
   int c[2] = {0};

   printf("Inside foo \n");
   printf("a[1] is: %d\n", a[1]);
   printf("b[1] is: %d\n", b[1]);
   printf("c[1] is: %d\n", c[1]);
}
```

```
indigo 308 % ./arrays
a[1] is: 0
b[1] is: 0
c[1] is: 0
Inside foo
a[1] is: 32562
b[1] is: 0
c[1] is: 0
indigo 309 % ./arrays
a[1] is: 0
b[1] is: 0
c[1] is: 0
Inside foo
a[1] is: 32536
b[1] is: 0
c[1] is: 0
```

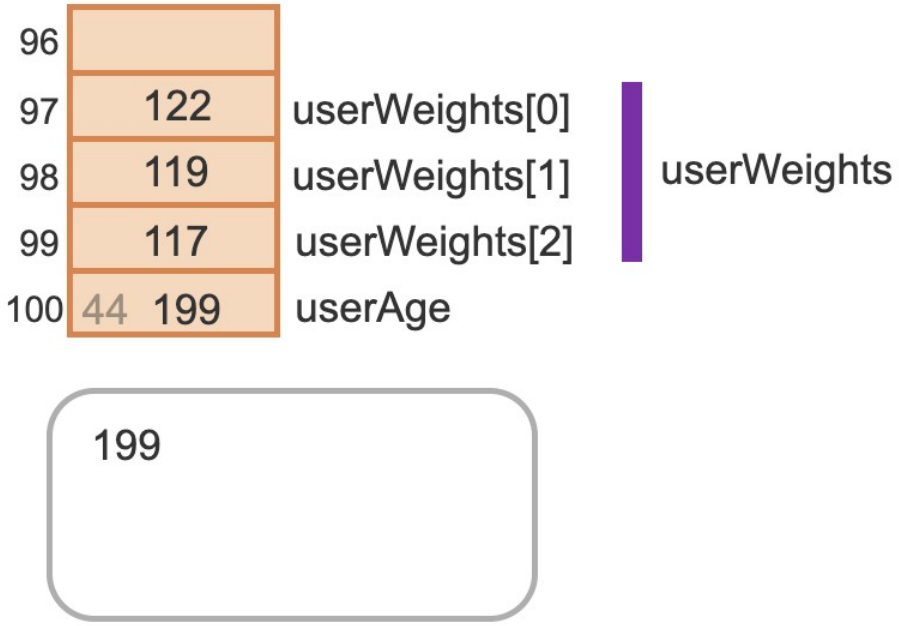
Iterating through arrays

```
int userWeights[3];
int userAge;

userAge = 44;

userWeights[0] = 122;
userWeights[1] = 119;
userWeights[2] = 117;
userWeights[3] = 199; // (Problematic)

// Print userAge
```



Two-dimensional arrays

```c
#include <stdio.h>

/* Direct driving distances between cities, in miles */
/* 0: Boston  1: Chicago  2: Los Angeles */

int main(void) {
   int cityA;                    // Starting city
   int cityB;                    // Destination city
   int drivingDistances[3][3]; // Driving distances

   // Initialize distances array
   drivingDistances[0][0] = 0;
   drivingDistances[0][1] = 960;  // Boston-Chicago
   drivingDistances[0][2] = 2960; // Boston-Los Angeles
   drivingDistances[1][0] = 960;  // Chicago-Boston
   drivingDistances[1][1] = 0;
   drivingDistances[1][2] = 2011; // Chicago-Los Angeles
   drivingDistances[2][0] = 2960; // Los Angeles-Boston
   drivingDistances[2][1] = 2011; // Los Angeles-Chicago
   drivingDistances[2][2] = 0;
```

Initializing a 2D array during the declaration

```
// Initializing a 2D array
int numVals[2][3] = { {22, 44, 66}, {97, 98, 99} };

// Use multiple lines to make rows more visible
int numVals[2][3] = {
   {22, 44, 66}, // Row 0
   {97, 98, 99}  // Row 1
};
```

Strings

- Single-quoted characters refer to numeric value in ASCII table

```
char x = 'A'; // same as x = 65
char y = '0'; // same as y = 48
```

- In C, strings are simply arrays of characters
  - Each element corresponds to a character
  - Character stored as numeric representation from ASCII table
  - Last character followed by a value of zero: termination byte

```
char s1[] = "EECS 2031";

char s2[] = { 'E', 'E', 'C', 'S', ' ', '2', '0', '3', '1', 0 };

char s3[] = {  69,  69,  67,  83,  32,  50,  48,  51,  49, 0 };
```

Char arrays / C strings

```c
#include <stdio.h>

int main(void) {
   char cityName[20] = "Forest Lake"; // Compiler appends null char

   // In each printf(), printing stops when reaching null char
   printf("%s\n", "City:");           // Compiler appends null char to "City:"
   printf("%s\n", cityName);

   return 0;
}
```

```c
int main(void) {
   char userStr[20] = "12345678901234567890"; // Input string
   int i;

   // Prompt user for string input
   printf("Enter string (<20 chars): ");
   scanf("%s", userStr);

   // Print string
   printf("\n%s\n", userStr);

   // Look for '@'
   for (i = 0; userStr[i] != '\0'; ++i) {
      if (userStr[i] == '@') {
         printf("Found '@'.\n");
      }
   }


   // The following is an ERROR.
   // May print chars it shouldn't.
   // Problem: doesn't stop at null char.
   printf("\n\""); // Print opening "
   for (i = 0; i < 20; ++i) { // Print each char
      printf("%c", userStr[i]);
   }
   printf("\"\n"); // Print closing "


   // The following is an even WORSE ERROR.
   // Accesses beyond valid index range.
   // Program may crash.
   printf("\""); // Print opening "
   for (i = 0; i < 30; ++i) {
      printf("%c", userStr[i]); // Print each char
   }
   printf("\"\n");               // Print closing "
```

```
Enter string (<20 chars): test@gmail.com

test@gmail.com
Found '@'.

"test@gmail.com6789"
"test@gmail.com6789$\305\366;\226\333"
```

An example involving reading char arrays

```
#include<stdio.h>
int length (char []);

main() {
    char my_strg[100];
    int a;

    printf("Enter a word and an int separated by blank>");
    scanf("%s %d", my_strg, &a);
    printf("%d %s %d", a, my_strg, length(my_strg));
}

int length(char arr[]){
     int i = 0;
     while (arr[i] != '\0')
       i++;
     return i;
}
```

No & needed!
Another big topic.
Investigate later

```
indigo 326 % a.out
Enter a word and an int by blank> hello 23
23 hello 5
```

Read string using scanf

```
char my_strg[100];
scanf ("%s", &my_strg);
scanf ("%s", my_strg);

printf("%s", my_strg);
```

String library functions

```
char orgName[100] = "United Nations";
char userText[20] = "UNICEF";
char targetText[10];
```

strcpy()	`strcpy(destStr, sourceStr)`
	
	Copies sourceStr (up to and including null character) to destStr.
		`strcpy(targetText, userText); // Copies "UNICEF" + null char`
		`// to targetText`
		`strcpy(targetText, orgName); // Error: "United Nations"`
		`// has > 10 chars`
		`targetText = orgName; // Error: Strings can't be`
		`// copied this way`

strncpy()	`strncpy(destStr, sourceStr, numChars)` Copies up to numChars characters.	`strncpy(orgName, userText, 6); // orgName is "UNICEF Nations"`
strcat()	`strcat(destStr, sourceStr)` Copies sourceStr (up to and including null character) to *end* of destStr (starting at destStr's null character).	`strcat(orgName, userText); // orgName is "United NationsUNICEF"`

strncat()	`strncat(destStr, sourceStr, numChars)`
	
	Copies up to numChars characters to destStr's end, then appends null character.
		`strcpy(targetText, "abc"); // targetText is "abc"`
		`strncat(targetText, "123456789", 3); // targetText is "abc123"`

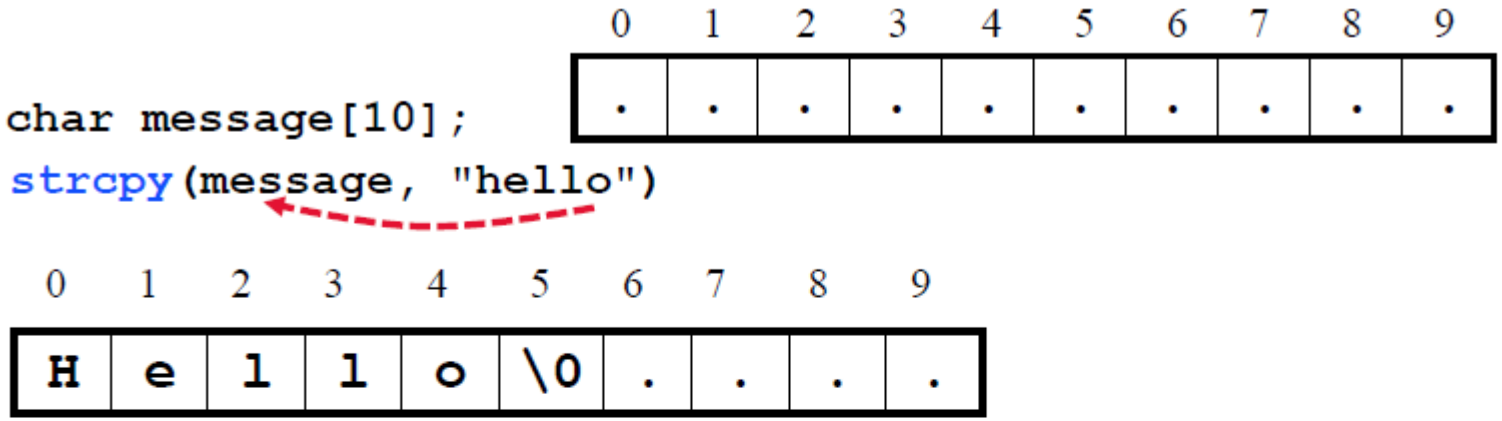
```
char orgName[100] = "United Nations";
char userText[20] = "UNICEF";
char targetText[10];
```

strchr()	`strchr(sourceStr, searchChar)`
	
	Returns NULL if searchChar does not exist in sourceStr. (Else, returns address of first occurrence, discussed elsewhere).
	NULL is defined in the string.h library.
		`if (strchr(orgName, 'U') != NULL) { // 'U' exists in orgName?`
		`  ...  // 'U' exists in "United Nations", branch taken`
		`}`
		`if (strchr(orgName, 'u') != NULL) { // 'u' exists in orgName?`
		`  ...  // 'u' doesn't exist (case matters), branch not taken`
		`}`

strlen()	`size_t strlen(sourceStr)`
	
	Returns number of characters in sourceStr up to, but not including, first null character. size_t is integer type.
		`x = strlen(orgName);  // Assigns 14 to x`
		`x = strlen(userText);  // Assigns 6 to x`
		`x = strlen(targetText); // Error: targetText may lack null char`

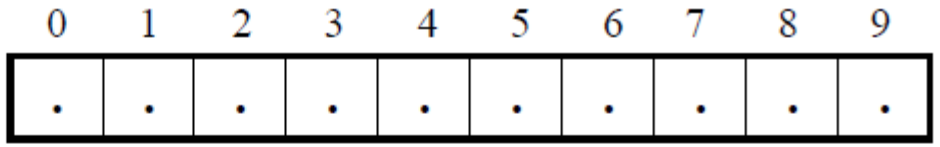
strcmp()	`int strcmp(str1, str2)`
	
	Returns 0 if str1 and str2 are equal, non-zero if they differ.
		`if (strcmp(orgName, "United Nations") == 0) {`
		`  ... // Equal, branch taken`
		`}`
		`if (strcmp(orgName, userText) != 0) {`
		`  ... // Not equal, branch taken`
		`}`

strcpy

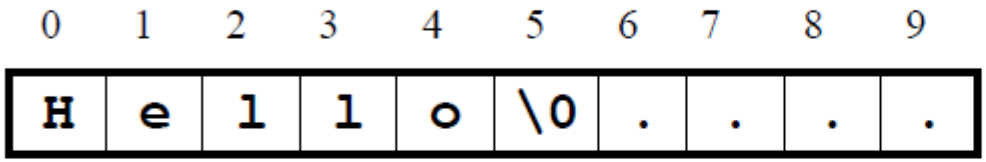



```
strlen(message)? 5   sizeof message? 10   message[4]? 'o'
printf("%s", message)?
```

strcpy

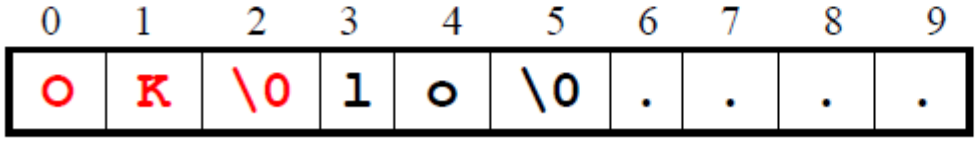



```
char message[10];
strcpy(message, "hello")
```



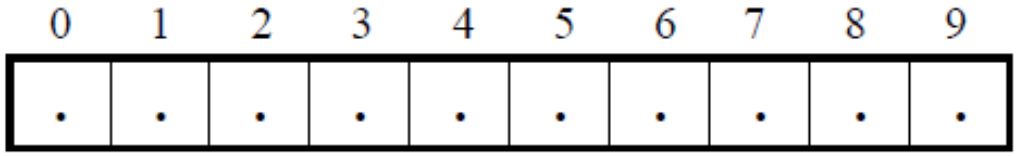

```
strlen(message)? 5   sizeof message? 10   message[4]? 'o'
printf("%s", message)?
```

```
strcpy(message , "OK");    ?
```

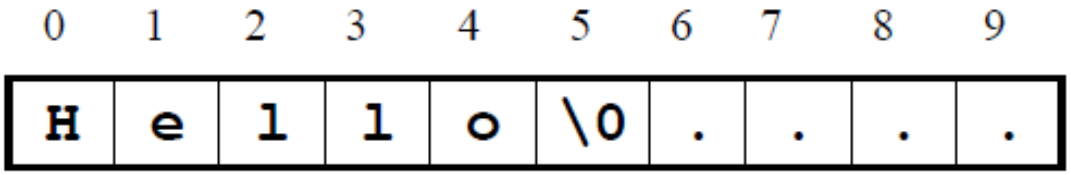



```
strlen(message)?     sizeof message?      message[4]?
printf("%s", message)?
```

strncpy

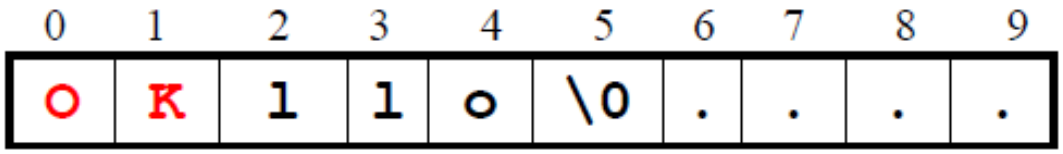



```
char message[10];
strcpy(message, "hello")
```




```
strlen(message)? 5   sizeof message? 10   message[4]? 'o'
printf("%s", message)?
```

```
strncpy(message , "OK",2);    ?
```



No \0 added

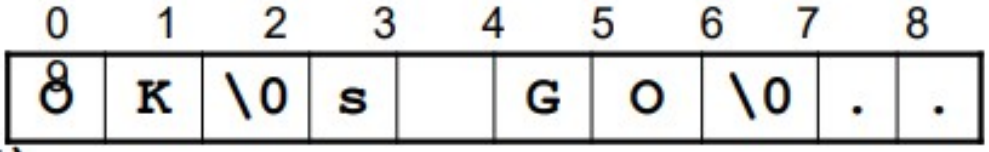
```
strlen(message)?     sizeof message?     message[4]?
printf("%s", message)?
```

strncpy

- What about *strncpy* (message, “ok”, 3)?

```
strlen(message)?    sizeof message?    message[4]?
printf("%s", message)?
```

strcat

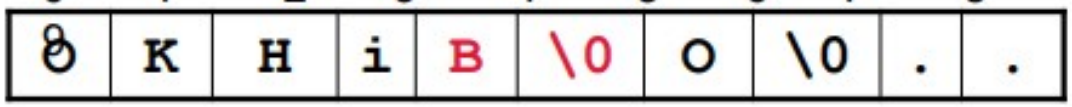



```
strcat(message, "Hi")
```

Append "Hi" to the end of message.
'H' replaces 1st '\0'

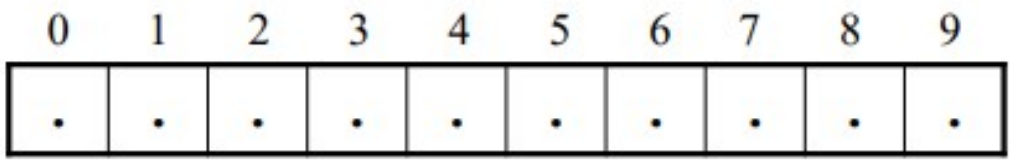
```
strlen(message)? 4   sizeof message? 10 message[6]? 'o'
printf("%s", message)?   OKHi
```

```
strcat(message , "B");    ?
```

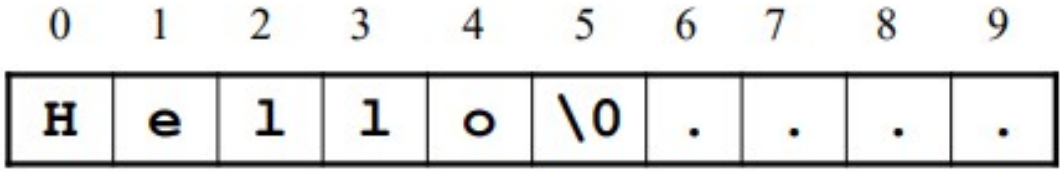



```
strlen(message)? 5   sizeof message? 10   message[6]? 'o'
printf("%s", message)? OKHiB        strncat(mes,"Bye",1)?
```

strcat

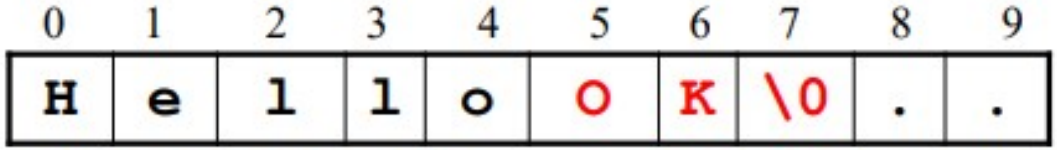



```
char message[10];
strcpy(message, "hello")
```



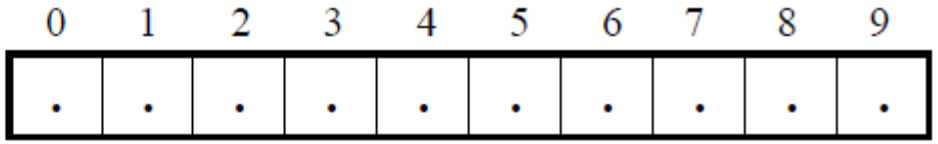

```
strlen(message)? 5   sizeof message? 10   message[4]? 'o
```

```
strcat(message , "OK");    ?          // Append "OK" to the end
                                      of message.
                                      'O' replaces 1st '\0'
```

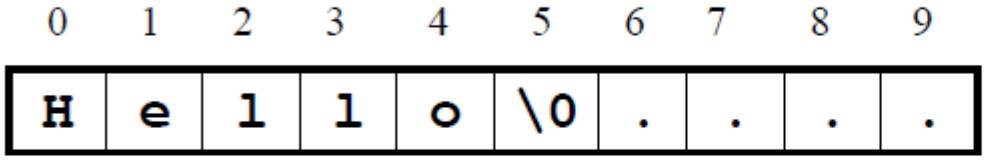



```
strlen(message)?     sizeof message?      message[5]?
printf("%s", message)?
```

strcat



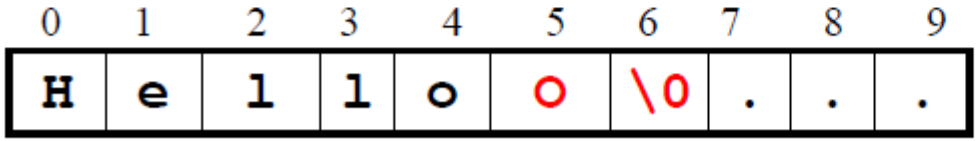

```
char message[10];
strcpy(message, "hello")
```




```
strlen(message)? 5   sizeof message? 10   message[4]? 'o'
```

```
strncat(message , "OK", 1);    ?
```

\0 added




```
strlen(message)?     sizeof message?     message[5]?
printf("%s", message)?
```

read with space in input?

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

int main() {
    char str[100] ;
    scanf("%s", str);
    printf("%s\n", str);
}
```

```
indigo 312 % getset
hello world
hello
indigo 313 % getset
i am
i
indigo 314 % getset
who you are
who
indigo 315 %
```

Inputting strings with white spaces

- **`fgets(str, num, stdin)`**

- Reads one line of characters from user input, ending with a newline, and writes those characters into the C string **`str`**.

- If a newline character is read from the user input before **`num`** characters are read, the newline character itself is also written into **`str`**, after which the function appends a null character.

- **`num`** is the maximum number of characters to be written into **`str`**.

- If **`num`** is 10 and the input line exceeds 10 characters, only the first 9 characters will be written into **`str`**, followed by the null character; the remaining input characters will not be read and will remain in user input.

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

int main(void) {
   char nameArr[10];        // User specified name
   char greetingArr[17];    // Output greeting and name

   // Prompt user to enter a name
   printf("Enter name: ");
   fgets(nameArr, 10, stdin);

   // Eliminate end-of-line char
   if (nameArr[strlen(nameArr) - 1] == '\n') {
      nameArr[strlen(nameArr)-1] = '\0';
   }


   // Modify string, hello + user specified name
   strcpy(greetingArr, "Hello ");
   strcat(greetingArr, nameArr);
   strcat(greetingArr, ".");

   // Output greeting and name
   printf("%s\n", greetingArr);

   return 0;
}
```

```
Enter name: Al Smith
Hello Al Smith.

...

Enter name: Mary Johnson
Hello Mary John.
```

Array of strings

```c
int main(void) {
   const int NUM_COUNTRY = 10;                                  // Number of countries supported
   const int MAX_COUNTRY_NAME_LENGTH = 50;                      // Max length for names
   char ctryNames[NUM_COUNTRY][MAX_COUNTRY_NAME_LENGTH]; // 2D array of country tv stats
   int arrPosition = 0;                                         // User specified position

   // Populate array
   strcpy(ctryNames[0], "U.S.A.");
   strcpy(ctryNames[1], "Italy");
   strcpy(ctryNames[2], "Poland");
   strcpy(ctryNames[3], "U.K.");
   strcpy(ctryNames[4], "Canada");
   strcpy(ctryNames[5], "Spain");
   strcpy(ctryNames[6], "France");
   strcpy(ctryNames[7], "Germany");
   strcpy(ctryNames[8], "Brazil");
   strcpy(ctryNames[9], "Russia");

   // Prompt user to enter desired position
   printf("Enter desired position (1-10): ");
   scanf("%d", &arrPosition);

   // Print results
   printf("People in %s watch the %d", ctryNames[arrPosition-1], arrPosition);
   if( arrPosition == 1 ) {
      printf("st");
   }
```

Math

- Defined in standard library, prototype in `<math.h>`
- Need to link by `-lm`

- `double sin(double x), cos(x), tan(x)`
- `double asin(x) acos(x) atan(x)` …
- `double exp(x)` e^x
- `double log(x) -- ln(x)`
- `double log10(x)`
- `double pow(x,y)` x^y
- `double sqrt(x)` $\sqrt{x}$
- `double ceil(x)` smallest int not less than x, as a double!
- `double floor(x)` largest int not greater than x, as a double!

x, y are of type double

return double

Char Classification

```
int islower(int ch)  ch >='a' && ch <='z'
int isupper(int ch)  ch >='A' && ch <='Z'
int isalpha(int ch)  islower(ch) || isupper(ch)
int isdigit(int ch)  ch >='0' && ch <='9'
int isalnum(int ch)  isalpha(ch) or isdigit(ch)
int isxdigit(int ch) '0'-'9', 'a'-'f','A'-'F',

int tolower(int ch)  if (isupper(ch))
int toupper(int ch)    return ch + ('a' - 'A');
                     else return ch;
```

assert.h

```
void assert(int expression)
```

```
int x = -1;
assert(x > 0)
```

print `Assertion failed: expression, file file, line lnum`

Then `abort()`

assert.h

Using the assert() macro.

```
1: /* The assert() macro. */
2:
3: #include <stdio.h>
4: #include <assert.h>
5:
6: main()
7: {
8:     int x;
9:
10:     printf("\nEnter an integer value: ");
11:     scanf("%d", &x);
12:
13:     assert(x >= 0);
14:
15:     printf("You entered %d.\n", x);
16:     return(0);
17: }
Enter an integer value: 10
You entered 10.
Enter an integer value: -1
Assertion failed: x, file list19_3.c, line 13
Abnormal program termination
```