CMPE-013/L

Introduction to "C" Programming

Max Lichtenstein

based on slides from Maxwell James Dunne

What will they print?

```
int i;= 1;
```

```
do {
    if (i == 15)
        break;
    i++;
    continue;
} while (0);

printf("i = %d\n", i);
```

```
int no_init;

printf("no_init = %x\n", no_init);
```

Announcements

- Lab 2 errors, please re-download zip
  - "board.h", RunCalculator()
  - Thanks anon!
- Piazza guidelines doc
- Pull-based assignment reports?

Roadmap

- Review of loops
- Review of conditionals (+ one new one)
- Functions
- I/O with scanf(), printf()
- Strings and other Arrays
- Some weird operators (?)

Loops Review

- 3 types:
  - `for( … ; … ;  … )  … ;`
  - `while ( … )  … ;`
  - `do  …  while … ;`
- And two control statements:
  - `continue;`
  - `break;`

Loops Review

- Loops can be nested, usually quite deep
  - (how deep can you go?)

```
char i, j, k;

for (i = 'a'; i <= 'c'; i++) {
    for (j = 'a'; j <= 'c'; j++) {
        for (k = 'a'; k <= 'c'; k++) {
            printf("%c%c%c\n", i, j, k);
        }
    }
}
```

9

a a a
a a b
a a c
a b a
a b b
a b c
a c a
⋮
c c b
c c c

- rs exist?

What does this output?

```
char i, j, k = 0;
    i = 'a'                                  b
~~for (i = 'a'; i <= 'c'; i++) {~~
    for (j = 'a'; j <= 'c'; j++) {
        for (i = 'a'; i <= 'c'; i++) {
            k++;
            printf("%d", k);
        }
    }
}
```

Conditionals Review

- 4 kinds:
  - `if ( … ) … ;`
  - `if ( … ) … else … ;`
  - `if ( … ) … else if ( …) … ;`
    - is this one really a different kind though?
  - and one new one…

Switch/Case

```
char x = 'b';

if (x == 'a') {
    printf("A");
} else if (x == 'b') {
    printf("B");
} else if (x == 'c') {
    printf("C");
} else if (x == 'd') {
    printf("D");
} else {
    printf("Not A,B,C, or D");
}
```

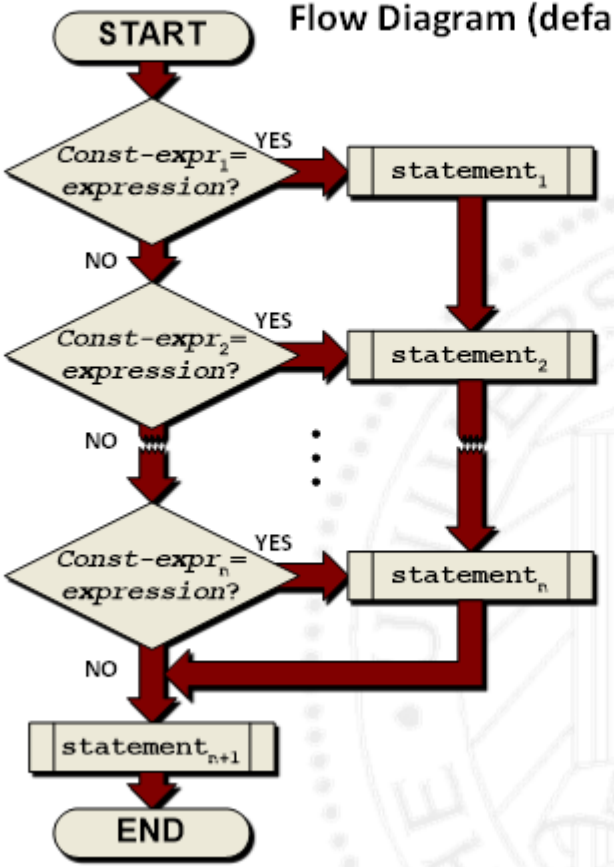
Switch/Case

```c
char x = 'b';

if (x == 'a') {
    printf("A");
} else if (x == 'b') {
    printf("B");
} else if (x == 'c') {
    printf("C");
} else if (x == 'd') {
    printf("D");
} else {
    printf("Not A,B,C, or D");
}
```

```c
char x = 'b';

switch(x){
case 'a':
    printf("A");
case 'b':
    printf("B");
case 'c':
    printf("C");
case 'd':
    printf("D");
default:
    printf("Not A,B,C, or D");
}
```

`switch` Statement

Flow Diagram (default)

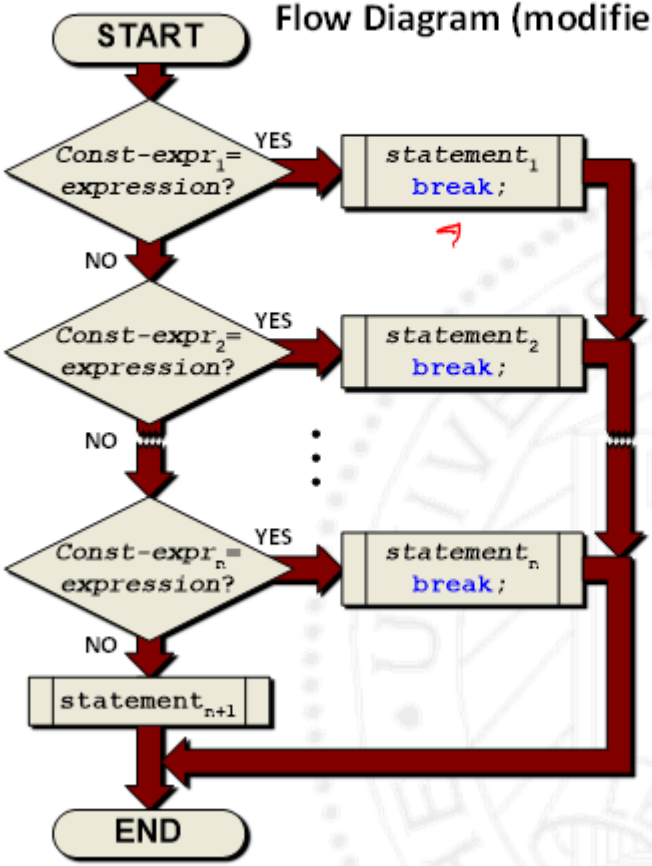


Notice that each statement falls through to the next

This is the default behavior of the `switch` statement

`switch` Statement

Flow Diagram (modified)



Adding a `break` statement to each statement block will eliminate fall through, allowing only one case clause's statement block to be executed

Functions

- What is a function?
- Reasons to use Functions
- How to use functions in C
- ~~Under the hood~~

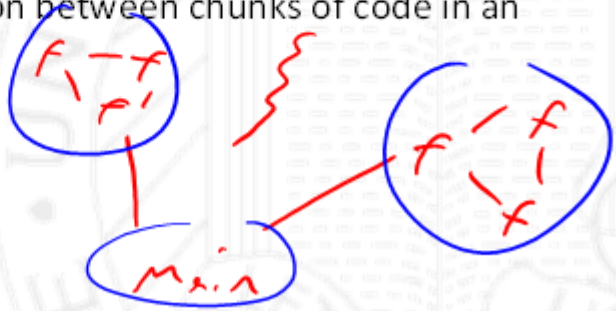
What is a Function?

Functions

Definition

Functions are self contained program segments designed to perform a specific, well defined task.

- All C programs have one or more functions
  - The `main()` function is required
- Functions can accept parameters from the code that calls them
- Functions return a single value (but can export more data)
- A way of moving information between chunks of code in an organized way
  - A *modular* way



Functions

Remember Algebra Class?

- Functions in C are conceptually like an algebraic function from math class…

Function Definition

Function Name → $f(x) = x^2 + 4x + 3$

Function Parameter

- If yo 7 gets "copied" into *x* and used everywhere that *x* exists within the function definition: $f(7) = 7^2 + 4*7 + 3 = 80$

Functions

Remember Algebra Class?

- When you call a function, the result gets "copied" in place of the call.

```
printf("%d\n",     f(7)     );

printf("%d\n", 7*7 + 4*7 + 3);

printf("%d\n",     80       );
```

Function Terminology

- Call:
  - Invoking the function
- Pass (or pass into):
  - To send information into the function
- Argument (or parameter)
  - The information you send in to the function*
    - *not the only way to get information in
- Return
  - The information that comes back from the function*
    - *not the only way to get information back

argument being passed

```
printf("%d\n",     f(7)     );
```

Call

```
printf("%d\n", 7*7 + 4*7 + 3);

printf("%d\n",      80      );
```

Return

Functions

Program Structure

```
main()
{
  ...
  eat();
  ...
  drink();
  ...
}
```

```
eat()
{
  ...
  return;
}
```

```
drink()
{
  ...
  be_merry();
  return;
}
```

```
be_merry()
{
  ...
  return;
}
```

Why use functions

- Reduce repetition
  - For coders and computers
- Hide information
- Organize and communicate concepts in your code
- Modularity
  - Share work from different sources
  - Test independently

Repetitiveness

```
//initialize LEDS:
LATECLR = 0xFF;
TRISECLR = 0xFF;

//Turn on LED 1,
LATE = 0x1;
//Delay for one second
for (i = 0; i < 1000000; i++);
//Turn off LED
LATE = 0;

//Turn on LED 2,
LATE = 0x2;
//Delay for one second
for (i = 0; i < 1000000; i++);
//Turn off LED
LATE = 0;

//Turn on LED 3,
LATE = 0x4;
//Delay for one second
for (i = 0; i < 1000000; i++);
//Turn off LED
LATE = 0;

///and on and on ....
```

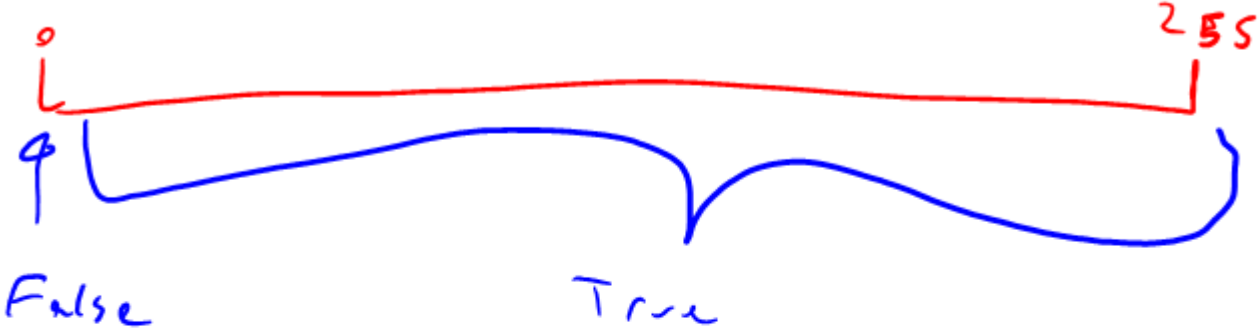
Information Hiding

BOARD_Init()

Organizing and Communicating Concepts

```
# define TRUE 1
# define FALSE 0
int i, j;

printf("2\n");
for (i = 3; i < 100; i++) {
    char isPrime = TRUE;
    for (j = 2; j < i; j++) {
        if (i % j == 0) {
            isPrime = FALSE;
            break;
        }
    }
    if (isPrime) {
        printf("%d\n", i);
    }
}
```

```
int i, j;

printf("2\n");
for (i = 3; i < 100; i++) {

    if (isPrime(i)) {
        printf("%d\n", i);
    }
}
```


0
255
False
True

Share Work / Modularity

```
printf("%d\n", i);
```

Using Functions in C

- Declarations and Prototypes
- Parameter
- Return values

Functions

Function Prototypes

- Just like variables, a function must be declared before it may be used
- Declaration must occur before main() or other functions that use it
- Declaration may take two forms:
  - The entire function definition
  - Just a function prototype – the function definition itself may then be placed anywhere in the program

Functions

Declaration and Use: Example 2

Example 2

```
int a = 5, b = 10, c;

int Maximum(int x, int y);

int main(void)
{
  c = Maximum(a, b);
  printf("The max is %d\n", c)
}

int Maximum(int x, int y)
{
  return ((x >= y) ? x : y);
}
```

Function is *declared* with prototype <u>before</u> use in main()

Function is *defined* <u>after</u> it is used in main()

Function Definiton

Syntax

Data type of return *expression*

Name

Parameter List (optional)

```
type identifier(type1 arg1,…,typen argn)
{
    declarations
    statements
    return expression;
}
```

int i;

Body

Header

Return Value (optional)

Void Functions

- The function type is `void` if:
  - The `return` statement has no *expression*
  - The `return` statement is not present at all
- This is sometimes called a *procedure function* since nothing is returned

Example

```
void identifier(type1 arg1,…,typen argn)
{
   declarations
   statements
   return;
}
```

`return;` may be omitted if nothing is being returned

Void Functions

- If no parameters are required, use the keyword `void` in place of the parameter list when defining the function

Example

```
type identifier(void)
{
      declarations
      statements
      return expression;
}
```

- What functions do you know that return no values?
  - i.e. return type void
- What functions do you know that take no parameters?
  - i.e. argument list void

Which Code is Valid?

```
float SquareRoot(float input);
int Square(int input);

int main(void){
    printf("%f", SquareRoot( Square(7) ) );
}
```

A

int

```
int SquareRoot(float input);
float Square(int input);

int main(void){
    printf("%d", SquareRoot( Square(7) ) );
}
```

B

Which Code is Valid?

```
int factorial(int x){
    return x * factorial (x-1);
}
```

< if (x==0) return 1;

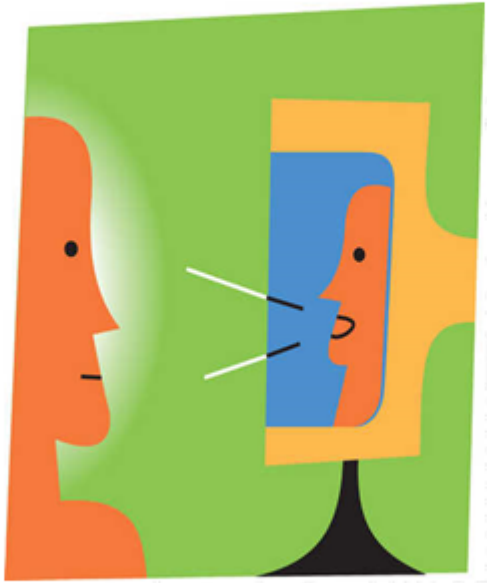
```
float polynomial( void ) {
    return 1*x*x + 4*x + 3 ;
}
```

B

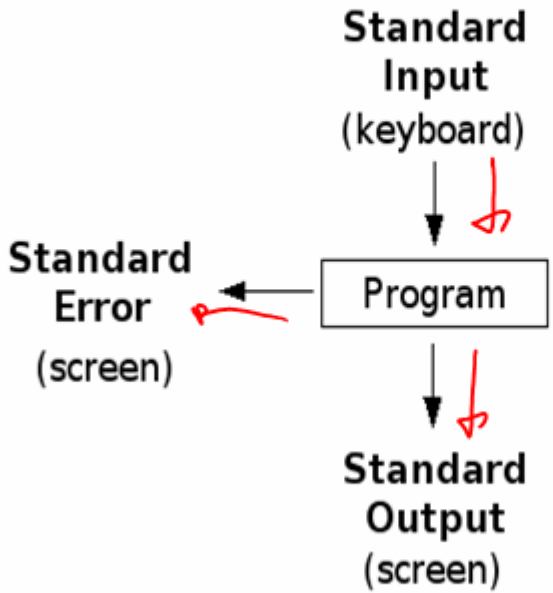
```
void polynomial( float x ) {
    return 1*x*x + 4*x + 3 ;
}
```

C

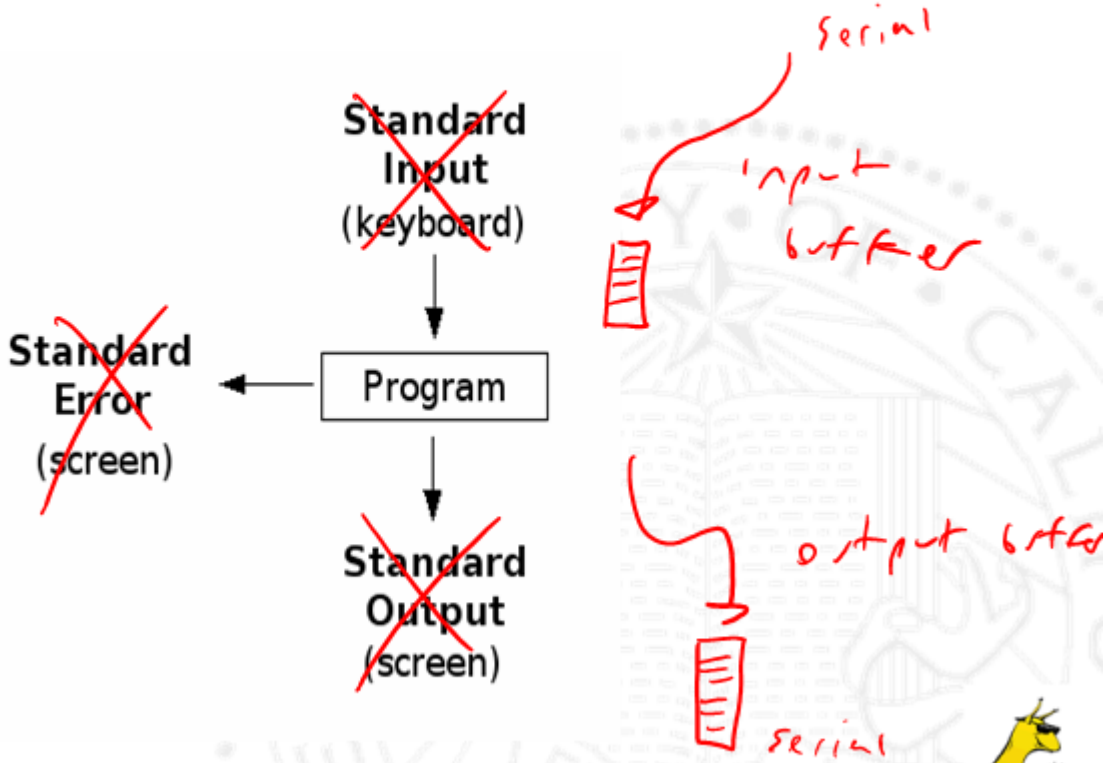
(Text) Input and Output



Standard Input and Standard Output

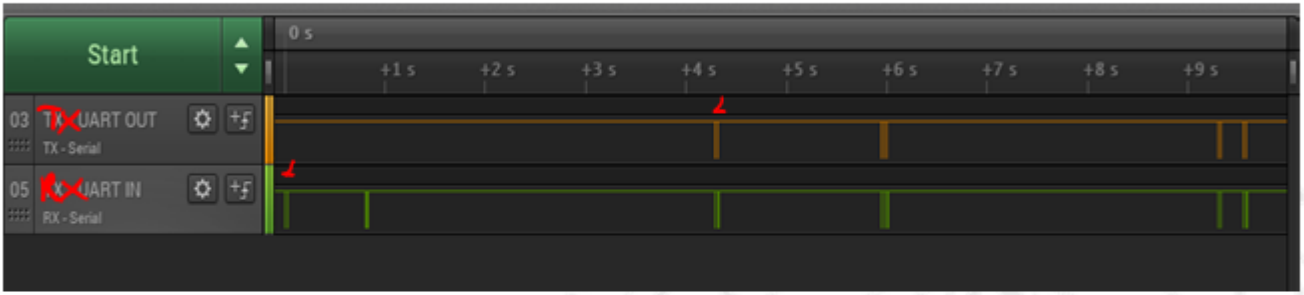


Embedded Input and Output

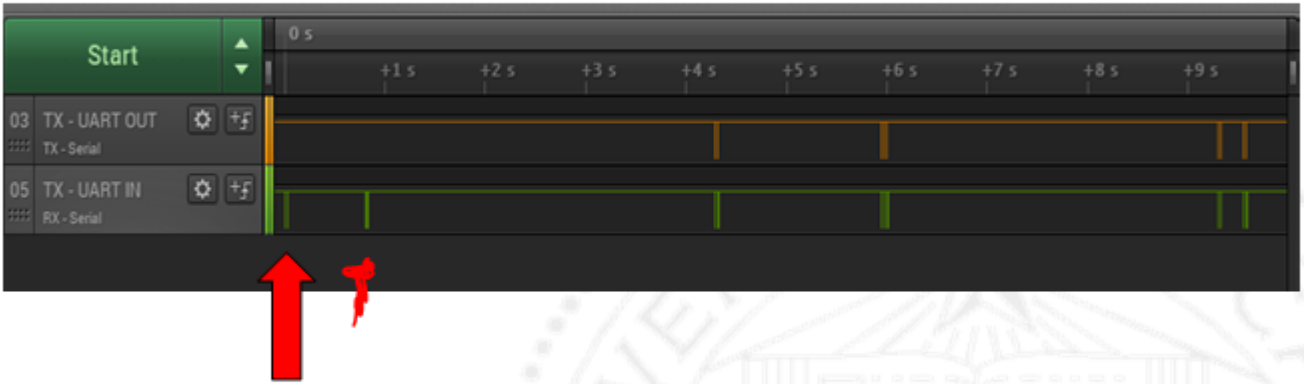


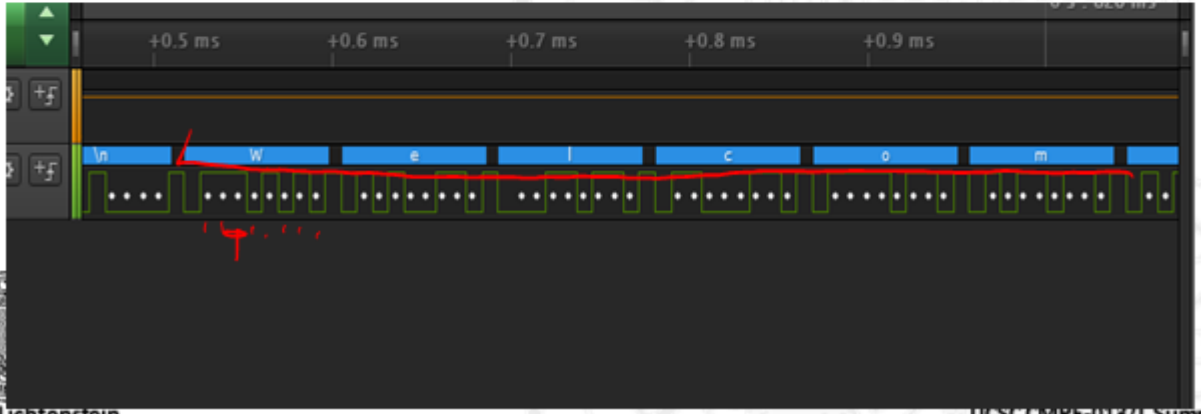
Input and Output Buffers

UART in Action

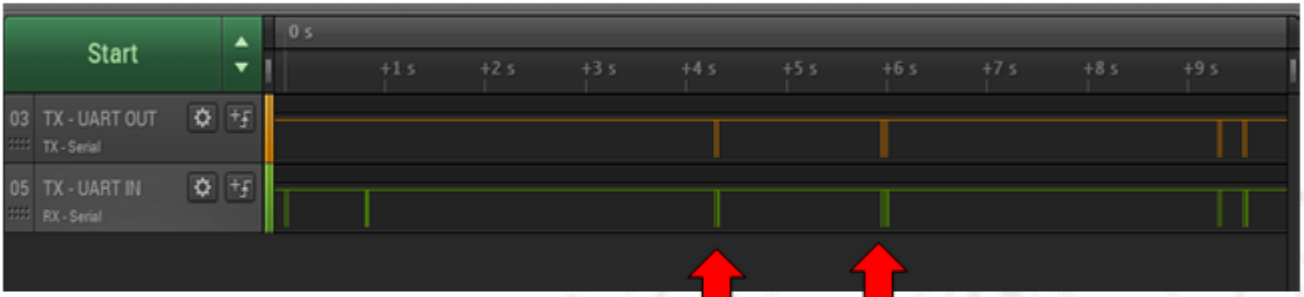


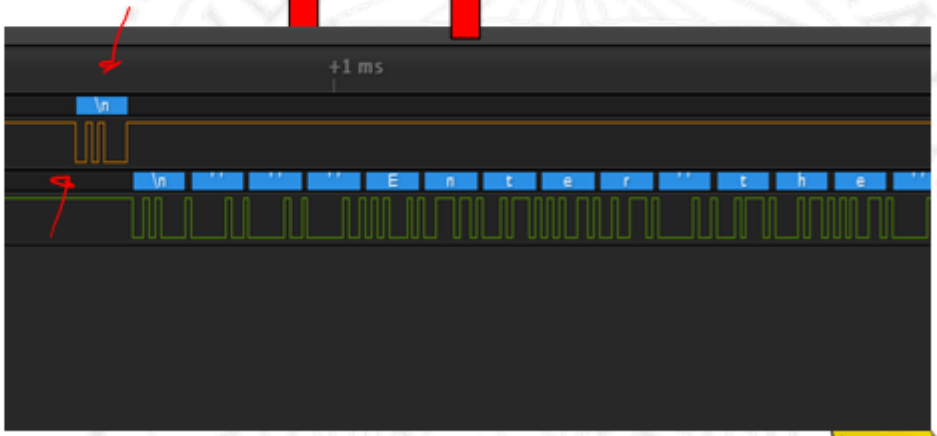
UART in Action





UART in Action



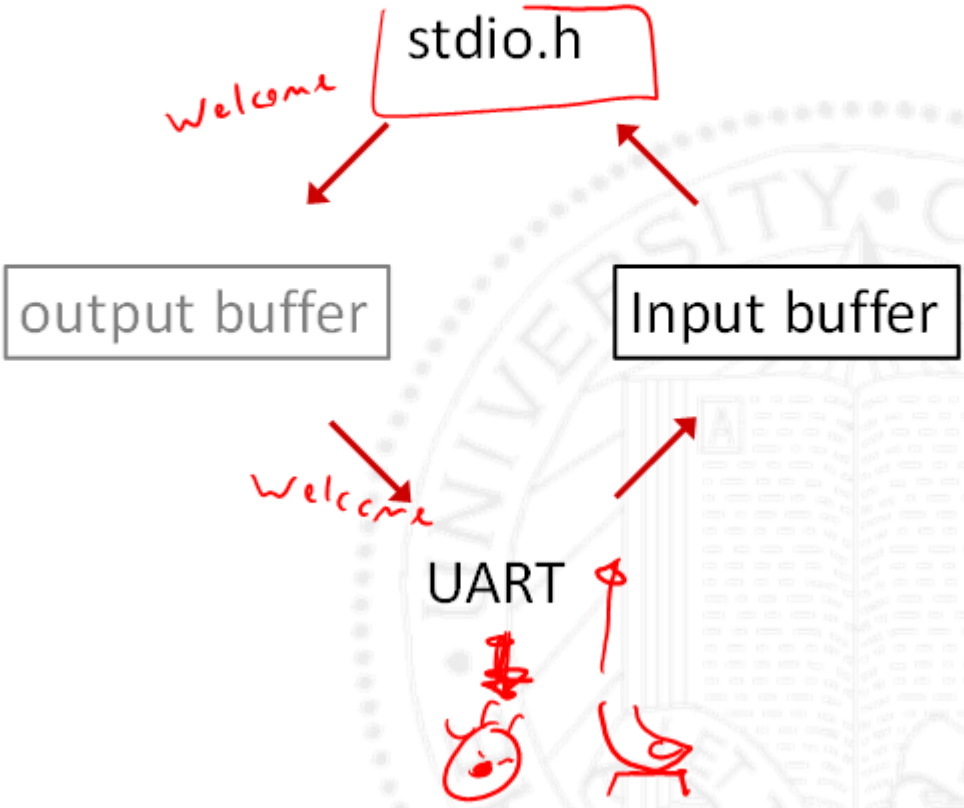


printf() and scanf()

- Standardized C tools for interacting with Standard Input and Standard Output
  - printf takes human unreadable binary data and turns it to human readable binary data (ascii)
  - ~~sprintf~~ scanf takes human readable binary data and turns it into machine unreadable binary data
- Defined in stdio.h
  - see also getc(), putc()

scanf()

The input buffer



printf()

Syntax

```
printf(ControlString, arg1, ..., argn);
```

- Everything printed verbatim within string except %d's which are replaced by the argument values from the list

```
int a = 5, b = 10;

printf("a = %d\nb = %d\n", a, b);
```

printf()

Format specifiers

%[flags][width][.precision][size]type

- Flags – Special printing options
- Width – The minimum size (in chars) of the output
- Precision – ~~Field width~~ # of digits after decimal point
- Size – Convert from base types to longer/shorter types
- Type – The base variable type

printf()

Format specifiers

```
float x = 0.5772156649015328606;

printf("%010.5f\n", x / 10000);
```

%[flags][width][.precision][size]type

What does it output?

```
float x = 0.5772156649015328606;
```

x/1000 = 0.9000577215665

```
printf("%010.5f\n", x / 10000);
```

0000.00006

More printf()

\	escape the next character
\\	print a backslash
"	start or end of string
\"	print a double quote
'	start or end a character constant
\'	print a single quote
%	start a format specification
\%	print a percent sign

%c	print a single character
%d	print a decimal (base 10) number
%e	print an exponential floating-point number
%f	print a floating-point number
%g	print a general-format floating-point number
%i	print an integer in base 10
%o	print a number in octal (base 8)
%s	print a string of characters
%u	print an unsigned decimal (base 10) number
%x	print a number in hexidecimal (base 16)
%%	print a percent sign (\% also works)

\a	audible alert (bell)
\b	backspace
\f	form feed
\n	newline (linefeed)
\r	carriage return
\t	tab
\v	vertical tab

flag	effect
none	print normally (right justify, space fill)
-	left justify
0	leading zero fill
+	print plus on positive numbers
␣	invisible plus sign

From Don Colton (BYU Hawaii),
http://www.cypress.com/file/54441/download

What does it print?

```
printf("Hello %s\n", "World");
```

Hello World

```
printf("%s", "Hello %s\n","World");
```

Hello %s

```
printf("%*s\n", 5, "123456789");
```

12345

```
printf("%c\n", "Hello World");
```

```
float a = 7;
printf("7 = %x\n", a);
```

0x 7

0x 3FF8700000

scanf()

Syntax

```
int scanf(FormatString, arg1, ..., argn);
```

- The format string tells *scanf* what kind of input.
- *arg1* through *argn* are <u>*POINTERS*</u> to variable of the right type.

```
int a, b;
printf("Input a and b\n");
scanf("%d %d", &a, &b);
```

input = 5 70

a = 5
b = 70

scanf()

Gotchas

scanf("␣%d");

scanf("␣");

- ~~Ignores blanks and tabs in format string~~
- Skips over white space (blanks, tabs, newline) as it looks for input
- Returns number of successful conversions
- Arguments ***must*** be pointers to variable types
- Arguments not processed in the input will be left in the input buffer.

scanf()

3	1	4	0		5	6	\n

scanf("%d %d", &a, &b)

3	1	4	0		5	6	\n

a = 3140, b = 56

scanf()

\n	7	7		-	3	\n

scanf("%d %d", &a, &b)

Nothing!

scanf()

3	1	4	0		5	6	\n

scanf("%d %d%c", &a, &b, &c)

3	1	4	0		5	6	\n

a = 3140, b = 56

sscanf.

scanf()

--size = 73

Format specifiers

scanf("%s %d", &x)

%[*][width][modifier]type

- * – Ignores this field
- Width – The maximum number of characters to match
- Modifier – Convert from base types to longer/shorter types
- Type – The base variable type

"%f"

("%lf", double)